Siebel Business Rules Administration Guide

Version 8.0, Rev. A

June 2007

Contents

1 What’s New in This Release

What’s New in Siebel Business Rules Administration Guide, Version 8.0, Rev. A

Table 1 lists changes described in this version of the documentation to support release 8.0 of the software.

Table 1. Documentation Changes in Siebel Business Rules Administration Guide, Version 8.0, Rev. A

Topic	Description
“Supported Installation and Uninstallation Scenarios” on page 31	Added requirement to run Object Importer in an existing knowledge base if Haley is uninstalled and re-installled.
“Writing Rules that Involve Parent and Child Business Components” on page 53	This added topic provides guidelines for writing rules on child business components.
“Using Some Standard Rule Constructs” on page 55	This added topic provides guidelines for avoiding errors when writing rules in some common scenarios.
“Setting the GetMoreData Input Property” on page 69 and “Setting the PerformAction Input Property” on page 70	These added topics provide requirements for setting the GetMoreData and PerformAction input properties for Business Rule Service.
“Scenario for Using Rules to Validate Data at Runtime” on page 78	This example now includes a rule module that is written on a child business component.

What’s New in Siebel Business Rules Administration Guide, Version 8.0

Siebel Business Rules Administration Guide and the functionality it documents are new in Oracle’s Siebel Business Applications release 8.0. Oracle’s Siebel Business Rules integrates Siebel Business Applications with the Haley inference engine and the HaleyAuthority rules authoring environment to allow you to develop and maintain business logic declaratively in an environment external to your Siebel applications.

2 About Business Rules

Siebel Business Rules allow the logic underlying business processes to be changed efficiently. This chapter provides the business case for implementing Siebel Business Rules.

The chapter contains the following topics:

About Siebel Business Rules Benefits

Changes in your business strategies result from changes in factors such as business conditions, inventory levels, seasons, and changes in personnel. Your CRM applications must adapt quickly to these changes and promptly reflect the changes in business priorities.

You can implement HaleyAuthority natural language business rules to automate decisions declaratively, instead of implementing those decisions programatically. By replacing scripts that can be difficult to develop and maintain with business rules that can be created and modified directly by business managers, the overall cost of maintenance and configuration of Siebel applications is minimized.

Some of the applicability of rules configurations in Siebel applications include:

- **Dynamic business processes.** Business processes are frequently dynamic; that is, their requirements and parameters can change on a regular basis. For example, insurance application approval, credit card approval, and insurance claim processing prerequisites can change over time. Siebel Business Rules allow such business process changes to be deployed declaratively in a natural language context, without the need for rewriting script logic or reconfiguring Siebel objects, and then recompiling the Siebel Repository.
- **Complex business processes.** Business processes, such as those you implement with Oracle's Siebel Workflow or Oracle's Siebel Task UI, frequently contain decision branching that depends on results of earlier steps. Rules allow more complex processing of data in order to generate more refined input values for decision-making. Changes to the business logic governing the input is maintained declaratively, instead of relying on entirely scripted solutions.
- **Field validations.** Besides easier implementation and maintenance during the life of a Siebel release, the basic field validations that are implemented with a rules engine are more easily and directly maintained across a Siebel upgrade than scripts that are used to do validations.
- **Auto-populating field values.** You can declaratively populate fields that depend on other field values. Rules allow changing the logic without having to recompile the Siebel Repository, and without having to modify script.

Implementing HaleyAuthority rules from a single source that is external to Siebel applications provides the following benefits:

- Groups of rules, or modules, for different purposes and applications can be developed and deployed separately:
  - Versioned objects can easily revert to previous versions.
  - Access levels to modules can be used to define who may modify particular rules and modules.
- Logic is more understandable when it exists as rules in natural language. Thus, participants other than software developers, such as business analysts and business managers, can create and modify the logic. The HaleyAuthority authoring tool provides visual cues to validate rules and parts of rules as they are written.
- A single rules engine allows you to maintain business logic in a single place for use with various Siebel applications and functionalities.
- When Siebel objects are imported to HaleyAuthority, all the necessary HaleyAuthority metadata constructs (concepts, relations, and phrasings) are created automatically. You can immediately begin writing rules to reflect your business logic.
- The HaleyAuthority test harness allows you to simulate rules execution before deployment or for doing regression testing.
- Rules are implemented as a proxy business service, and can be invoked with the following mechanisms:
  - Runtime events
  - Siebel workflows
  - Siebel Task-based UI
  - Business services

  NOTE: The same business logic (rule module) can be reused in any of these contexts.
- Rules can invoke:
  - Siebel business component methods
  - Application methods
- Siebel object model changes in the Siebel Repository and the imported HaleyAuthority constructs are synchronized on demand.

About Using Business Rules Documentation

This guide uses the term *rules* in two ways:

- Business rules is the familiar connotation for well-defined business processes.
- Rules are also the logical statements constructed in HaleyAuthority that capture and implement business processes in Siebel applications. Typically in this document, mention of creating, deploying, and implementing rules refers to HaleyAuthority rule statements.

You have the following sources of documentation for implementing and using Haley Rules Engine with Siebel objects:

- *HaleyAuthority Help* documentation is available through HaleyAuthority's Help menu. It includes documentation on the local machine and it also links to Haley's Web site. HaleyAuthority documentation is the primary source for instruction on building rules in HaleyAuthority, including information about:
  - HaleyAuthority entities
  - Procedures for building rules
  - Testing and debugging rules
  - Examples of rules
- *Siebel Business Rules Administration Guide* is the primary source for information on implementing HaleyAuthority rules with Siebel objects, including information about:
  - Examples of rules in various Siebel application contexts, including runtime events, Siebel workflows, Siebel Task-Based UI flows, business services, and script
  - Examples of rules using various Siebel objects
  - Implementing and using Siebel Object Importer
  - Using the Deployment plug-in
  - Installing Haley Rules Engine and integrating Haley Rules Engine with Siebel applications
  - The Siebel applications, Haley Rules Engine, and HaleyAuthority integrated architecture
  - Processes in the development life cycle
  - Rules development access control
  - Synchronizing the Siebel Repository objects with the HaleyAuthority knowledge base
  - Best practices for writing and deploying rules with Siebel objects and applications
  - Localizing rules
  - Considerations specific to using Haley Rules Engine with Siebel objects and applications
  - Importing and exporting rules
  - Maintaining and administering rules
  - Troubleshooting rules
- This document is the secondary source, to Haley documentation, for information including:
  - Best practices for creating rules
  - Rules terminology
- See *Siebel Master Data Applications Privacy Guide* for information about prepackaged rules for Oracle's Siebel Master Data Applications.

3 About Developing and Deploying Siebel Business Rules

This chapter provides an end-to-end overview of implementing a business process that leverages rules in your Siebel application. The chapter contains the following topics:

- "Design Process Overview" on page 13
- "Development Process Overview" on page 19

Design Process Overview

This process includes the following tasks that are typically performed by a business analyst or business developer. Depending on your project, you may perform one or more of these tasks. Typically, you perform these tasks in the order they are presented here.

1. "Defining the Business Process Flow" on page 13
2. "Specifying Business Logic Requirements" on page 14
3. "Specifying Business Process Invocation Requirements" on page 15
4. "Defining UI Elements for Interactive Business Processes" on page 16
5. "Determining Siebel Development Tools for Implementing the Business Process" on page 17

Defining the Business Process Flow

This topic is a task in the process "Design Process Overview" on page 13.

In this task you summarize the business process that you want to implement. This task is an initial definition of requirements.

Activities

- State the actions to be taken.
- State requirements for the actions.
- State conditions that support requirements.
- Plan the flow of the process.
- Determine dependencies.
- Answer questions about the nature of the process:
  - Does the process require user interactivity?
  - Does the process validate user input?
  - Does the process modify data?
  - Does the process make decisions based on values of data?
  - Is the process multi-step?
  - Are steps in the process dependent on each other?
  - Is this process a batch process?

For information, see:

HaleyAuthority User Guide

"Scenario for Using Rules to Validate Data at Runtime" on page 78

"Scenario for Validating Data Using Script to Invoke Rules" on page 94

"Scenario for Using Rules in a Siebel Workflow to Do Batch Processing" on page 104

"Scenario for Using Rules to Provide Dynamic Navigation in a Siebel Task" on page 128

Specifying Business Logic Requirements

This topic is a task in the process "Design Process Overview" on page 13.

In this task you sharpen the definition of the business process.

Activities

- Work backward to more clearly define the business process policies:
  - State the actions to be taken.
  - For each action, state the direct requirements (conditions that must be satisfied) for the action to be taken.
    - Identify conditions that need only to be satisfied individually in order for the action to be taken.
    - Identify conditions that must be satisfied in combination with other conditions in order for the action to be taken.
    - Identify exceptions to other conditions.
  - In turn, state conditions and exceptions that support higher level conditions.
  - Identify assumptions
- Sharpen the business process policy definitions.
  - Clearly state the individual conditions that make up compound conditions.
  - Minimize negative logic when possible.
  - Define limits and ranges.

For information, see: *HaleyAuthority User Guide*

"Scenario for Using Rules to Validate Data at Runtime" on page 78

"Scenario for Validating Data Using Script to Invoke Rules" on page 94

"Scenario for Using Rules in a Siebel Workflow to Do Batch Processing" on page 104

"Scenario for Using Rules to Provide Dynamic Navigation in a Siebel Task" on page 128

Specifying Business Process Invocation Requirements

This topic is a task in the process "Design Process Overview" on page 13.

In this task you determine how and when the business process starts.

Activities

The following guidelines suggest Siebel development tools that meet invocation requirements:

- The user manually invokes the process:
  - Siebel Task UI and Siebel Workflow allow the user to invoke processes.
  - Script can underlie UI buttons.
    - Script can leverage rules.
- The process is invoked by a runtime event:
  - Siebel runtime events can be used to trigger processes.
    - Runtime events can leverage rules, script, or both.
    - Siebel workflow processes can be invoked in response to Siebel runtime events.
- The process is invoked on a preset schedule:
  - Siebel workflow processes, such as periodic batch processes, can be invoked on a preset schedule.

For information, see:

"Scenario for Using Rules to Validate Data at Runtime" on page 78

"Scenario for Validating Data Using Script to Invoke Rules" on page 94

"Scenario for Using Rules in a Siebel Workflow to Do Batch Processing" on page 104

"Scenario for Using Rules to Provide Dynamic Navigation in a Siebel Task" on page 128

Defining UI Elements for Interactive Business Processes

This topic is a task in the process "Design Process Overview" on page 13.

In this task you specify the UI messages and dialog boxes that are needed for user interaction, and you design input fields, labels, and buttons.

Activities	■ Plan the text for static messages. ■ Messages defined in rule modules typically do not require design work. ■ For multi-step tasks, design the layout, including text boxes, buttons, other controls, and the control labels. ■ Decide how user actions populate record fields. ■ Provide controls that gather user input that is required as criteria for process decisions. ■ If text in UI elements must be localized: ■ Leverage language independent code (LIC) in your Siebel environment.
For information, see:	"Scenario for Using Rules to Validate Data at Runtime" on page 78 "Scenario for Using Rules to Provide Dynamic Navigation in a Siebel Task" on page 128

Determining Siebel Development Tools for Implementing the Business Process

This topic is a task in the process "Design Process Overview" on page 13.

In this task you determine which Siebel development tools can be leveraged to implement the business process. The rules engine is one of the tools that is available.

Many processes that can be implemented by using the rules engine can also be implemented using script. However, rules allow much of the logic of the process to be maintained declaratively and external to your Siebel application. Changing the logic maintained in the rules does not incur downtime because no recompiling of the Siebel Repository is needed.

NOTE: This design task is not included in the examples in this document. The development tool in each example fits the business process and invocation requirements.

Activities In addition to invocation requirements, the following guidelines suggest Siebel development tools that fit various requirements of the business process, and aspects that may suggest implementing the rules engine:

- The user interacts in the process:
  - Siebel runtime events can be used in response to specific events.
    - Rules engine can be used to implement more complex tests on field values and on calculations, and can display messages for different conditions.
    - Rules engine can be used to set multiple field values that are dependent on more complex criteria.
  - Siebel Task UI can be used to implement multiple-step tasks that are initiated by the user.
    - Rules engine allows complex decision-making and task-branching that depend on output from rules.
    - Rules engine allows client-side validation based on UI events, such as entering a text field, to be embedded in the task.
  - Siebel Workflow can be used to implement multiple-step tasks that can be initiated in response to events.
    - Rules engine allows complex decision-making and task-branching that depends on output from rules.
- The process validates data:
  - Some fundamental data validation can be implemented using business component user properties or field user properties.
  - Validation of data entry by the user can be implemented with Siebel runtime events, such as PreWriteRecord, or with Siebel Workflow.
  - Non-interactive data validation can be implemented using runtime events, such as WriteRecord, or with Siebel Workflow for seamless multi-step processing, such as batch processing.
  - For runtime event or Workflow implementations, rules engine can be used to perform more complex decision-making and to set field values that depend on more complex criteria. With Siebel workflow, rules output can be used as criteria for process branching.
- Other non-interactive workflows, such as batch processing, can also be implemented by Siebel Workflow.

For information, see:

“Scenario for Using Rules to Validate Data at Runtime” on page 78

“Scenario for Using Rules in a Siebel Workflow to Do Batch Processing” on page 104

“Scenario for Using Rules to Provide Dynamic Navigation in a Siebel Task” on page 128

Development Process Overview

This process includes the following tasks that are typically performed by a Siebel configuration developer. Depending on your project, you may perform one or more of these tasks.

1. “Adding Business Logic” on page 19
2. “Testing Rule Modules in HaleyAuthority” on page 21
3. “Deploying and Activating Rule Modules” on page 21
4. “Creating the Business Process Vehicle” on page 22
5. “Providing Calls to the Rules Engine” on page 23
6. “Deploying the Business Process” on page 24
7. “Testing the Business Process” on page 24
8. “Migrating the Business Process Between Environments” on page 25
9. “Administering the Business Process” on page 25

The order that these tasks are performed can vary from project to project. Some tasks can be done interchangeably for a project. For example, creating rules in HaleyAuthority and creating a Siebel Task UI flow in Siebel Tools are independent tasks and may be done in either order.

Adding Business Logic

This topic is a task in the process “Development Process Overview” on page 19.

You implement rules to make decisions and, if applicable, take action. To construct rules from business logic, you first identify the actions or decisions that are the final product of the business logic, then work backward to construct the conditions to evaluate and entities and relations on which the conditions are built.

Activities

Translate the logic requirements you derived in "Specifying Business Logic Requirements" on page 14 into modules of rule statements, using the natural language conventions of the HaleyAuthority authoring environment.

- Write rule statements to implement the actions that are the end products of the logic; for example, set the value of a field, display a message, set an output property value, and so on.
- For each action statement, write applicability statements (conditions) that trigger the action.
    - Write conditions that suffice to trigger the action individually as *if* applicability statements. These conditions are evaluated as *or* conditions.
    - Write conditions that must be satisfied, independent of other conditions, as *only if* applicability statements. These conditions are evaluated as *and* conditions.
    - Write conditions that negate the trigger, independent of any other conditions, as *except* applicability statements. These conditions are evaluated as *and not* conditions.
  - Similarly, state conditions and exceptions that support higher level conditions.
  - Write statements for assumptions, if necessary.
- Sharpen the structure of the statements in each module.
  - Applicability statements supporting a higher level statement are typically easier to understand and debug than creating a complete if-then condition as one statement.
    - Avoid compound conditions in an individual statement by expressing the conditions as individual applicability statements.
  - Minimize negative logic.

For information, see:

HaleyAuthority User Guide is the primary source of instruction on creating rule modules.

"Creating Rule Modules" on page 52

"Scenario for Using Rules to Validate Data at Runtime" on page 78

"Scenario for Validating Data Using Script to Invoke Rules" on page 94

"Scenario for Using Rules in a Siebel Workflow to Do Batch Processing" on page 104

"Scenario for Using Rules to Provide Dynamic Navigation in a Siebel Task" on page 128

Testing Rule Modules in HaleyAuthority

This topic is a task in the process "Development Process Overview" on page 19.

You can test your rule modules with simulated data in HaleyAuthority before deploying them to the runtime database.

Activities

- Create test cases in HaleyAuthority
- Create simulated data - instances and facts - in HaleyAuthority and associate them with test cases.
- Run test cases.
- Examine test results
- Modify data and rerun test cases.

For information, see: *HaleyAuthority User Guide* is the primary source of instruction on testing rule modules using the test harness.

"Using the HaleyAuthority Test Harness" on page 56

Scenario for Using Rules to Validate Data at Runtime

Scenario for Using Rules in a Siebel Workflow to Do Batch Processing

Deploying and Activating Rule Modules

This topic is a task in the process "Development Process Overview" on page 19.

You must deploy a rule module to write it to the runtime database. You must activate the rule module to make it available to use.

Activities

- Deploy a rule module in HaleyAuthority.
- Activate a rule module in your Siebel application's Business Rules Administration screen.

For information, see: "Deploying Rule Modules" on page 59

"Configuring and Activating a Deployed Rules Module" on page 63

"Scenario for Using Rules to Validate Data at Runtime" on page 78

"Scenario for Validating Data Using Script to Invoke Rules" on page 94

"Scenario for Using Rules in a Siebel Workflow to Do Batch Processing" on page 104

"Scenario for Using Rules to Provide Dynamic Navigation in a Siebel Task" on page 128

Creating the Business Process Vehicle

This topic is a task in the process "Development Process Overview" on page 19.

In this task, you use Siebel development tools to create the structure in which the business process is implemented. Depending on the vehicle you choose, you do activities in the following areas:

- "Creating a Siebel Runtime Event" on page 22
- "Creating a Siebel Task" on page 22
- "Creating a Siebel Workflow" on page 23

Creating a Siebel Runtime Event

You can implement rules in supported business component runtime events to do tasks such as field validation and dynamic setting of field values, in which it is not necessary to access the content in the output property set that results from calling the rule.

Activities

- Configure an action set to invoke the Business Rule Service business service with appropriate rule modules.
- Associate the action set with a runtime event that is supported by the rules engine.

For information, see: *Siebel Personalization Administration Guide*

"Integrating Rules with a Siebel Runtime Event" on page 75

Creating a Siebel Task

Siebel Task UI allows you to define user-interactive tasks that help the user to complete business processes.

Activities

- Create the task.
  - Layout the task flow in the Task Editor.
  - Set properties for the task, the task steps, and the step connectors.
  - To implement rules in the task, you must include a business service step.
- Create business component and task group definitions.
  - Define the views in which the task is available to the user
- Create UI elements.
  - For a task UI, this involves creating task views.
- Bind the task views to task steps.

For information, see: *Siebel Business Process Framework: Task UI Guide*

"Implementing a Rules Module in a Siebel Task UI Flow" on page 127

Creating a Siebel Workflow

Siebel Workflow allows you to define multi-step processes that can run on a pre-defined schedule, in response to a runtime event, or when manually initiated. A Siebel workflow process may or may not be user-interactive.

Activities

- Create the workflow process in the Process Designer.
  - Layout the workflow.
  - Set properties for the workflow process, the workflow process steps, and the step connectors.
  - To implement rules in the workflow process, you must include a business service step.
- Create UI elements.
  - For interactive workflow processes, you can add synthetic event controls to existing views to enhance process navigation.

For information, see:

Siebel Business Process Framework: Workflow Guide

"Implementing a Rules Module in a Siebel Workflow" on page 103

Providing Calls to the Rules Engine

This topic is a task in the process "Development Process Overview" on page 19.

If you implement rules in a Siebel Task, a Siebel Workflow, or in script, you must build a scripted call to the Business Rule Service business service. The script provides the input properties to Business Rule Service, and, if necessary, parses its output property set.

The basic structure of the script is the same in all cases.

Activities

To implement rules in a Siebel Task UI or Workflow:

- Create a custom business service that consists of script that makes a call to the Business Rule Service business service.
  - Build the script to parse the output property set of Business Rule Service if output from the rules is input to other steps.
- Create a business service step that invokes the custom business service.
- To pass output from the business service step to other steps, translate the parsed output property set into task properties for a task or process properties for a workflow process.

You can use script in other contexts to implement rules. The script must call the Business Rule Service business service.

For information, see:	“Implementing a Rules Module to Execute From Script” on page 93
	“Scenario for Using Rules in a Siebel Workflow to Do Batch Processing” on page 104
	“Scenario for Using Rules to Provide Dynamic Navigation in a Siebel Task” on page 128

Deploying the Business Process

This topic is a task in the process “Development Process Overview” on page 19.

The processes for deploying Siebel runtime events, workflow processes, tasks, and scripts does not vary from their standard deployment processes because the vehicle includes calls to the rules engine.

Deploying your business process includes tasks such as publishing, activating, and configuring, depending on the vehicle that you choose to implement the process. These tasks are done in Siebel Tools or in administration screens in your Siebel application.

For information, see:	*Siebel Business Process Framework: Workflow Guide*
	Siebel Business Process Framework: Task UI Guide
	Siebel eScript Language Reference
	Siebel Personalization Administration Guide
	“Scenario for Using Rules to Validate Data at Runtime” on page 78“Scenario for Validating Data Using Script to Invoke Rules” on page 94
	“Scenario for Using Rules to Provide Dynamic Navigation in a Siebel Task” on page 128

Testing the Business Process

This topic is a task in the process “Development Process Overview” on page 19.

You can test the business process in a development runtime environment before deploying the process to the production environment. For some deployment vehicles, you can also do simulation testing.

Calls to the rules engine do not change the strategies for testing business processes implemented with Siebel runtime events, workflow processes, tasks, and scripts.

For information, see:	*Siebel Business Process Framework: Workflow Guide*
	Siebel Business Process Framework: Task UI Guide
	Siebel eScript Language Reference
	"Scenario for Using Rules to Validate Data at Runtime" on page 78"Scenario for Validating Data Using Script to Invoke Rules" on page 94
	"Scenario for Using Rules in a Siebel Workflow to Do Batch Processing" on page 104
	"Scenario for Using Rules to Provide Dynamic Navigation in a Siebel Task" on page 128

Migrating the Business Process Between Environments

Migration of Siebel runtime events, workflow processes, tasks, and scripts does not vary because the vehicle includes calls to the rules engine. However, migration of rules runtime data and your HaleyAuthority knowledge base between environments does require additional tasks.

You migrate the business process from your development environment to the production environment, and you likely will migrate your process from one or more development environments to the primary development environment.

Activities

- Standard migration activities for runtime events, workflow processes, tasks, and scripts

Rules-specific activities include:

- Migrate knowledge base data between environments
- Migrate rules runtime data between environments.

For information, see:

"Migrating Rules Between Environments" on page 39

Siebel Application Deployment Manager Guide

Administering the Business Process

This topic is a task in the process "Development Process Overview" on page 19.

Any change in your business process or in its invocation strategy may require you to do administration activities. These changes can result from events such as the following:

- The flow of your process changes.
- The parameters of your business logic change; for example, thresholds that generate actions change.
- You modify the rule statements in the process.

- Your Siebel Repository changes; for example, you upgrade your Siebel applications release.
- You wish to deactivate the process temporarily or permanently.

Activities	■ Standard administration activities for runtime events, workflow processes, tasks, and scripts Rules-specific activities include: ■ Synchronize the knowledge base with the Siebel Repository. ■ Reconfigure rules. ■ Deactivate rules.
For information, see:	“Synchronizing the HaleyAuthority Knowledge Base with the Siebel Repository” on page 50 “Reconfiguring and Deactivating Rule Modules” on page 65 *Siebel Database Upgrade Guide* *Siebel Business Process Framework: Task UI Guide* *Siebel Business Process Framework: Workflow Guide*

4 About Siebel Business Rules Architecture

This chapter provides information on the business rules architecture. The chapter contains the following topics:

About the High Level Rules Architecture

Figure 1 shows the high-level rules architecture.

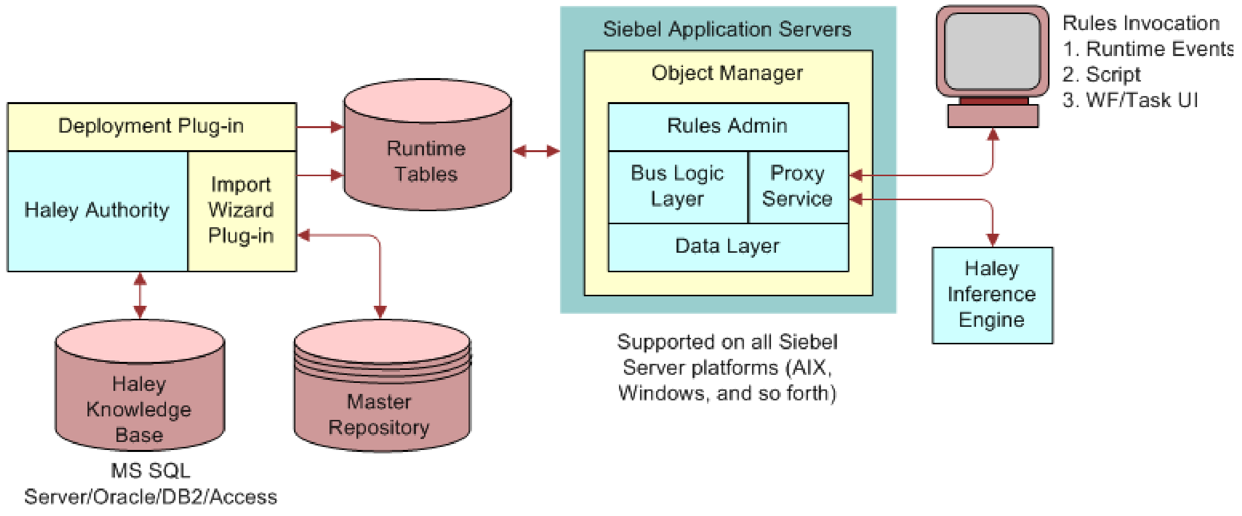


Figure 1. High-Level Rules Architecture

Rules that are implemented in Siebel Business Applications are authored in Haley Enterprise's HaleyAuthority.

In order for users to write rules on Siebel objects, HaleyAuthority must have knowledge of the Siebel objects. Results of executed rules map to operations on Siebel objects in Siebel applications.

The following components of the rules architecture allow rules authored in HaleyAuthority to execute in Siebel applications:

- **Haley Enterprise's HaleyAuthority** is a rule-authoring tool in which you define rules.

  For more information, see "About HaleyAuthority" on page 28.

- **Siebel Object Importer** (Import Wizard in Figure 1 on page 27) is a plug-in to HaleyAuthority. Using Object Importer, you select a subset of the Siebel object model and import it into HaleyAuthority to create the concepts and relations that are needed to define rules on Siebel objects.

  For more information, see "About Siebel Object Importer" on page 29.

- **Siebel Deployment plug-in** is a plug-in to HaleyAuthority. Using the Deployment plug-in, rule modules created in HaleyAuthority are compiled and stored into runtime tables. For deployment across multiple Siebel servers, the Application Deployment Manager (ADM) framework is used.

  For more information, see "About Siebel Deployment Plug-In" on page 30.

- **Rules Runtime Administration screen** (Rules Admin in Figure 1 on page 27) is used to activate, deactivate, and configure deployed rule modules.

  For more information, see "About Siebel Rules Runtime" on page 30.

- **HaleyAuthority knowledge base** stores HaleyAuthority's metadata, including Object Importer-created Siebel object definitions and rules.

- **Haley inference engine** executes rules.

- **Proxy Service** is Siebel's Business Rule Service business service. It is a proxy to the rules engine functionality at runtime. It manages Haley's runtime objects, manages deployed rule modules, interfaces with the Application Object Manager for data, interfaces with Haley's inference engine for rule execution, and manages the rule execution results from Haley's inference engine.

For information on the data flow among the HaleyAuthority knowledge base, the Siebel Master Repository database, and the Siebel runtime database, see "Understanding Building and Deploying Rules" on page 33.

About HaleyAuthority

HaleyAuthority is set of tools for modeling business policies as rule statements in English, without the need to employ programming languages. HaleyAuthority allows you to test, implement, and deploy rule statements. The statements are executed by Haley's inference engine.

Through integration with Siebel applications, the executed rules can be used to act on data or make decisions in Siebel application processes.

Thus, HaleyAuthority allows business logic to be maintained declaratively and in a location external to your Siebel implementation.

Haley documentation, included in HaleyAuthority through the Help menu, provides the primary instruction for using HaleyAuthority to build generic rules.

For information about using this Siebel documentation with Haley documentation, see "About Using Business Rules Documentation" on page 10.

About Siebel Object Importer

Before you can start to write a rule, HaleyAuthority must be made aware of the things on which the rule operates.

The HaleyAuthority knowledge base includes a semantic role model. In short, the HaleyAuthority semantic role model includes the following basic structural components:

- Concepts play roles in relations.
- Relationships are instances of relations. Roles in a relationship are filled by instances of the concepts that play the corresponding roles in the base relation. For example, the relation *A person has a mother* could have a relationship *Joe's mother is Lee*.
- Determiners and quantifiers are supported on roles in relationships within sentences.
- Relations may include verb phrasings.
- Concepts may include nouns or noun phrasings, including adjectives.

For detailed information on the HaleyAuthority semantic role model, see the Haley documentation.

To write rules that operate with Siebel objects, Siebel object definitions must first be mapped to HaleyAuthority's semantic role model. Siebel Object Importer Plug-in is a utility that is integrated with HaleyAuthority and does the mapping on demand.

With Siebel Object Importer, you can select and import into HaleyAuthority the minimal set of business components and fields to write a particular rule or set of rules.

Siebel Object Importer is used to perform any of the following tasks:

- Importing Siebel object definitions by helping a user introspect the Siebel object model to pick only the subset of objects that is needed to define rules. When the user picks a business object all of the business components belonging to the business object are listed. The user can drill down and pick individual business components and fields.
- Synchronizing the HaleyAuthority knowledge base with the Siebel Master Repository; that is, making the HaleyAuthority concepts and relations that represent imported Siebel objects consistent with the current Siebel object model
- Initializing a new knowledge base by adding a set of concepts, relations, procedures, verbs, and verb phrasings that are required for implementing rules in Siebel applications

When you import Siebel objects into HaleyAuthority, Siebel Object Importer builds the following HaleyAuthority objects:

- HaleyAuthority concepts that represent Siebel business components and fields
- HaleyAuthority relations that are generated from links, or 1:M relations, between Siebel business components. Phrasings are generated for such relations, such as *An expense has an expense item*.
- HaleyAuthority relations that are generated between Siebel business components and child fields. Phrasings are generated for such relations, such as *An account has a name*.

About Siebel Deployment Plug-In

The Deployment plug-in lists top level rule modules from HaleyAuthority, lets users select modules to be deployed, compiles selected rule modules and deploys them into the Siebel runtime environment. It gives users an option to not deploy the semantic role model.

Rule modules can be deployed without having to recompile the Siebel Repository or restart the Siebel Server.

About Siebel Rules Runtime

Siebel Rules Runtime (or simply Siebel runtime) provides the communication mechanism between Haley Runtime and the Application Object Manager. Siebel Rules Runtime performs the following:

- Runtime Administration:
  - Provides configuration, activation, and inactivation of deployed rule modules.
  - Provides functionality for importing and exporting rule modules.
- Rules Invocation:
  - Integrates the Haley inference engine with the Application Object Manager.
    - Passes Siebel runtime data to the Haley inference engine.
    - Manages Haley's runtime knowledge base (an Application Object Manager instantiation), which it loads with rule modules.
    - Manages working memory which is used to load runtime data that the rule module is run against.
    - Handles invocations into the Haley inference engine.
    - Performs rule execution results from the Haley inference engine in the Siebel application.
    - Provides tracing and logging.
- Haley Inference Engine Extension:
  - Extends Haley Inference Engine to pass back the results to the Siebel application.
  - Performs currency amount comparison operation callbacks.
  - Performs Siebel extended action, predicate, and function callbacks.
  - Provides tracing and logging.

5 Installing the Business Rules Infrastructure

This chapter provides guidelines for installing and uninstalling HaleyAuthority. The chapter contains the following topic:

Supported Installation and Uninstallation Scenarios

This topic describes the installation and uninstallation scenarios that are supported by Oracle Corporation for implementation of HaleyAuthority rules with Siebel applications.

You must observe the following general guidelines for setting up Siebel applications to implement HaleyAuthority rules:

- HaleyAuthority may only be installed once for each machine. If you attempt to install Siebel Tools and HaleyAuthority on a machine that already has HaleyAuthority installed, the installation appears to succeed, but the second HaleyAuthority instance actually fails silently and is not installed.
- In a single development or production environment, consisting of Siebel Tools, Siebel Master Repository, and Siebel database, you must have no more than one instance of HaleyAuthority and one instance of the HaleyAuthority knowledge base.

HaleyAuthority is Windows-compatible only and is installed by default with Siebel Tools.

CAUTION: To implement HaleyAuthority rules with Siebel applications, you must install and uninstall HaleyAuthority only as part of the Siebel Tools install and uninstall processes.

By default, HaleyAuthority is installed in the \RULE subdirectory of the Siebel Tools root directory.

CAUTION: Do not change the name of the \RULE subdirectory during installation or after installation. However, you can opt to change the Siebel Tools root directory during installation, but not after installation.

The \RULE\Documents subdirectory contains Haley documentation for using HaleyAuthority, including the HaleyAuthority User's Guide. You can access the User's Guide from HaleyAuthority's Help menu.

Two program groups are created for HaleyAuthority in Start > Programs: one as Haley Systems at the top level of the hierarchy, and one as Siebel Business Rules Developer at the same level as Siebel Tools.

HaleyAuthority appears as a standalone program in Windows' Control Panel's Add/Remove programs. However, users must never uninstall HaleyAuthority separately from Siebel Tools. Instead, uninstall Siebel Tools, which includes uninstallation of HaleyAuthority. If you uninstall HaleyAuthority separately, then HaleyAuthority cannot be re-installed separately and work with Siebel applications. Instead, you must then uninstall Siebel Tools, then re-install Siebel Tools, including HaleyAuthority.

NOTE: If you uninstall Haley, then reinstall Haley (all as part of uninstalling and installing Siebel Tools) and then open an existing knowledge base, you will generate an error if you attempt to deploy or test rule modules. To avoid the error, you must first run Object Importer to recreate the siebel.dat file, which stores various information about the knowledge base and runtime database.

Check your Control Panel's Add/Remove programs list for any independent listing of HaleyAuthority only as a precaution after uninstalling HaleyAuthority as part of a Siebel Tools uninstall. Remove any instance of HaleyAuthority that you find.

NOTE: Even if Haley provides a new release of HaleyAuthority, you must only install HaleyAuthority as part of the Siebel Tools installation in order for HaleyAuthority to work with Siebel applications. Siebel's Object Importer plug-in is tied to a particular release of HaleyAuthority, so a new HaleyAuthority release is not typically compatible with Siebel applications.

If you must have multiple instances of Siebel Tools installed on a machine, as in a development environment, then you must choose a custom installation of Siebel Tools and explicitly uncheck the Siebel Business Rules Developer checkbox for all but one of the Siebel Tools installations.

For detailed information about installing and uninstalling Siebel applications, see *Siebel Installation Guide* for the operating system you are using.

6 Configuring the Business Rules Development Environment

This chapter provides instruction on configuring and maintaining your rules development environment. The chapter contains the following topics:

Understanding Building and Deploying Rules

In order to invoke rules in a Siebel application, the following events take place:

- You build a HaleyAuthority knowledge base by importing Siebel object metadata from a Siebel Master Repository database and creating rule modules.
- You deploy rule modules to a Siebel runtime database.
- The rules are invoked at Siebel application runtime.

Figure 2 represents the interaction among the HaleyAuthority knowledge base, Siebel runtime database, and Siebel Master Repository database when rules are created, deployed, and invoked.

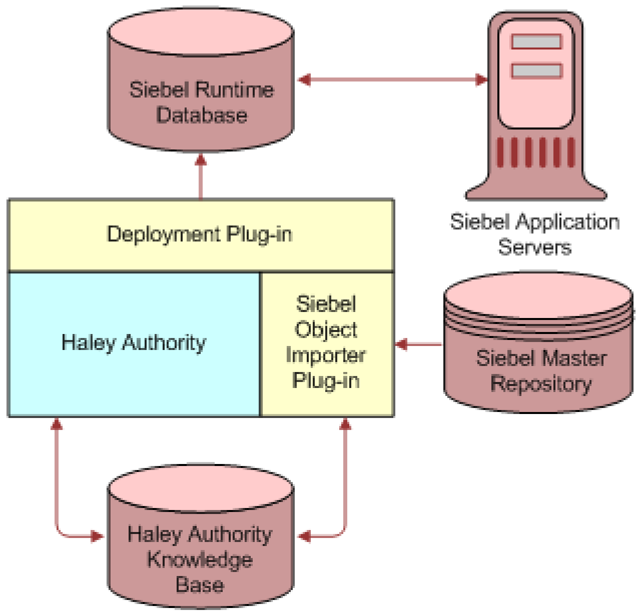

Siebel Runtime Database
Siebel Application Servers
Deployment Plug-in
Haley Authority
Siebel Object Importer Plug-in
Siebel Master Repository
Haley Authority Knowledge Base

Figure 2. Database Interaction for Rules Creation and Deployment

The following three databases participate in implementing rule modules.

- **Siebel Master Repository database.** The Siebel Master Repository database must contain the most current Siebel object model for your enterprise, from which you import Siebel objects into HaleyAuthority. The Master Repository database may or may not contain transaction data. The Master Repository database in a development environment is the database to which you check in repository changes using Siebel Tools.
- **Siebel runtime database.** A Siebel runtime database is a database with transaction data and seed data (lists of values, price lists, and so on). It must have the same repository data as your Master Repository database. A Siebel runtime database and the corresponding Siebel Master Repository database may or may not be the same database. In a primary development environment, they are likely separate databases for security and performance reasons.
- **HaleyAuthority knowledge base.** A HaleyAuthority knowledge base is the rules design environment. It is modified directly by HaleyAuthority and consists of the Haley schema definition and Haley data - concepts, relations, statements and so on. A knowledge base must not reside in the Siebel runtime database or in the Siebel Master Repository database.

  NOTE: Haley's runtime knowledge base is neither the Siebel runtime database nor the HaleyAuthority knowledge base. The runtime knowledge base is a representation of the semantic role model and rules that is instantiated in the Application Object Manager. For more information on Haley's runtime knowledge base, see "About Rule Modules Invocation: Loading, Using, and Caching" on page 36.

The following processing occurs when you create, deploy, and invoke rule modules.

About Rule Modules Creation

Whenever you use Object Importer, a COM Data Control (CDC) connection to a Siebel Master Repository database is created and a CDC connection to a Siebel runtime database is created.

When you import Siebel objects using Object Importer:

- Siebel object metadata is read from the Master Repository database.
- When Object Importer has the final list of objects to import, it tells HaleyAuthority, through the Haley API, to create concepts, relations, and phrasings in the HaleyAuthority knowledge base.
- Concurrently, Object Importer writes data to the Siebel runtime database that keeps track of the business components and fields that are imported. This data allows knowledge base data to be interpreted correctly by the Siebel runtime database at runtime.

  NOTE: When you import objects the first time into a knowledge base, Object Importer verifies that the runtime tables do not contain pre-existing data, such as from another knowledge base. If data is found, a message warns you that the runtime tables will be cleared. When you affirm, the runtime tables are cleared. Do not affirm unless you want to abort any existing knowledge base that accesses the runtime knowledge base.

As expected, the length of time to import objects depends on the number of objects you import.

When you subsequently create statements and rule modules, they are written to the HaleyAuthority knowledge base.

For information about tables provided for rules data in the Siebel runtime database, see "Rule Tables in the Siebel Runtime Database" on page 183.

For detailed information about importing Siebel objects and creating rules, see "Using Siebel Object Importer" on page 47 and Chapter 7, "Creating and Deploying Rules."

About Rule Modules Deployment

When you deploy rule modules from HaleyAuthority:

- The Siebel Deployment plug-in compiles the HaleyAuthority object model (semantic role model) and each selected rule module in the HaleyAuthority knowledge base.
- The Deployment plug-in updates the runtime database by inserting or modifying the runtime tables with the compiled binaries.

For information about tables provided for rules data in the Siebel runtime database, see "Rule Tables in the Siebel Runtime Database" on page 183.

For detailed information about deploying rule modules, see "Deploying Rule Modules" on page 59.

About Rule Modules Invocation: Loading, Using, and Caching

You can implement rule modules to execute in various scenarios; for example, in a runtime event or in a business service within a Siebel workflow. Independent of how a rule module is invoked, the activity that occurs consists of loading the binaries (object model and rule modules) that exist in the runtime database into the Haley's runtime knowledge base for execution.

Haley's runtime knowledge base is instantiated in the Application Object Manager when the object manager process starts, and persists until the object manager process stops.

NOTE: Haley's runtime knowledge base is neither the Siebel runtime database nor the HaleyAuthority knowledge base. For more information on the Siebel runtime database and the HaleyAuthority knowledge base, see "Understanding Building and Deploying Rules" on page 33.

A rule module is loaded into Haley's runtime knowledge base for execution when the rule module is first invoked. Haley's runtime knowledge base is loaded with rule modules on demand and shared among different user sessions. Once a rule module is loaded into the Haley's runtime knowledge base, it is ready to be used until it needs to be excised.

When a rule module is re-deployed or deactivated, the loaded rule module is excised from Haley's runtime knowledge base. The next time the rule module is invoked, the newly deployed or re-activated rule module is loaded into the runtime knowledge base.

Managing Users and Groups in HaleyAuthority

By default, HaleyAuthority has three user groups - Administrator, Everyone, and Owners. You can control access by individual users in a group by assigning privileges and permissions to the group.

Each group has a set of privileges that define high level access control. By default, a member of any of these groups has all privileges. The privileges include:

- Take ownership
- Change privileges
- Generate logic
- Change default permissions
- Administer users and groups
- Rollback or commit
- Change workflow states
- Change custom fields

In addition, permissions can be assigned to each group to define access control at the object level. A permission restricts or allows each member of a group to perform certain actions on particular object types or on individual objects. For example, users in the Everyone group may be disallowed from deleting relations or disallowed to delete and individual relation.

For an object that is a child of another object, the permissions on the parent object are inherited by the child.

The following notable principles apply to users of a knowledge base:

- The network user (the Windows network login) who launches HaleyAuthority and defines the new knowledge base automatically becomes a member of the Administrators group in the new knowledge base.
- Any user (Windows network login) who opens the knowledge base automatically becomes a member of the Everyone group.
- Any user who runs the object importer and, as a result, a new HaleyAuthority entity, value, or relation is created, becomes the owner of the created objects. Privileges assigned to owners of objects are administered in the Owners group.
- By default, a member of any group can create new groups and can change the privileges of any group.

It is recommended that an appropriate administrator change the privileges of the groups to limit actions of individuals as your organization prefers. For example, it is not advisable for everyone to be able to redefine privileges for others, so you may want to change the default behavior by deleting the *administer users and groups* and *change privileges* privileges from the Everyone and Owners groups and leave those privileges to Administrators only.

CAUTION: For restrictions on HaleyAuthority knowledge base modifications that are required for Oracle support, see "This appendix provides technical support guidelines and troubleshooting information. The appendix contains the following topics:" on page 193.

For detailed information on the following topics, and on other HaleyAuthority security topics, see *HaleyAuthority Help.*

About Rules Development Environment Topology

HaleyAuthority versions statements, so changes can be tracked. However, HaleyAuthority and Siebel plug-ins do not include a check-out/check-in system to keep changes by multiple developers cumulative.

The recommended development model is to maintain one knowledge base for creating and deploying rules to a single runtime database.

- The production knowledge base must only be used to update the production runtime database, and the production database must be updated by one and only one knowledge base.
- A development knowledge base must only be used to update one and only one development runtime database, and the development database must be updated by one and only one knowledge base.
- Individual developers must replicate all changes to the development knowledge base that is tested in the development runtime database before migrating to production.
- Individual developers can use a local knowledge base for prototyping only. As in the production and primary development environments, the local knowledge base must update one and only one runtime database (likely a local database), and the runtime database must be updated by one and only one knowledge base.

- All development on local knowledge bases must be replicated on the development knowledge base.

Setting Up Your Knowledge Base

You can create a knowledge base in an existing database or as a new Access database.

- An Access knowledge base (AKB) must only be used for local prototyping. It is not intended to be the knowledge base in a production, development, or test environment.
- It is strongly recommended that you provide a dedicated database for any production, development, or testing knowledge base.

CAUTION: Do not allow a knowledge base to share a Siebel runtime database or a Siebel Master Repository database. Neither of these configurations is supported.

Setting up a knowledge base consists of the following tasks:

- Creating a Database to Store the Knowledge Base, for most implementations
- Defining Connectivity to the Knowledge Base

Creating a Database to Store the Knowledge Base

For a production, development, or testing implementation, create a separate database with an enterprise relational database management system such as Oracle, MSSQL, or IBM DB2. Configure the an ODBC datasource to this database.

Defining Connectivity to the Knowledge Base

When you launch HaleyAuthority, you must specify the knowledge base to use for the session. If you create a new knowledge base, then you must define connectivity to its datasource.

You can create a new knowledge base in an existing ODBC datasource or as a new Microsoft Access database. You must only use an Access knowledge base for local prototyping.

If you create your new knowledge base in an existing ODBC datasource then you must first:

- Confirm that your client has the required ODBC drivers.
  - The Siebel Tools installation process configures the ODBC for the Siebel Repository. If your HaleyAuthority knowledge base uses a different RDBMS than the repository, then you must provide the required ODBC driver for the client.
- Create the ODBC datasource for the HaleyAuthority knowledge base.

To define connectivity for a new knowledge base for a production, development, or testing implementation

1. Launch HaleyAuthority in either of the following ways:
   - Choose Start > Program Files > *Program Group that contains Siebel Tools* > Siebel Business Rules Developer.

- Choose Start > Program Files > Haley Systems > HaleyAuthority Enterprise.

2. After launching HaleyAuthority, the HaleyAuthority dialog box displays. Under Create a new knowledge base using, click the Existing ODBC datasource radio button, then click OK.

   NOTE: Although a local Access knowledge base can be used for a single developer's prototyping, the developer may also choose to set up a non-Access knowledge base for the same purpose.

3. In the New Knowledge Base dialog box, in the ODBC datasources and drivers tab, select the ODBC datasource for the new database that you created to store the knowledge base, then click OK.

4. On the following dialog box, enter the login credentials for the datasource, and click OK.

   NOTE: For this first user who is creating the HaleyAuthority knowledge base in the existing datasource, the login must have the *create table* privilege, otherwise HaleyAuthority cannot create the new knowledge base. This is the only time that the *create table* privilege is required of a user of this knowledge base.

   HaleyAuthority creates its tables and initializes the knowledge base.

After a knowledge base in an ODBC datasource is created, any user who launches HaleyAuthority can open the knowledge base by using the *Open an existing knowledge base* option in the HaleyAuthority dialog box. If the ODBC datasource is not in the list, select *More datasources* and select the ODBC datasource from the resulting list. The user must provide the ODBC login credentials to use the knowledge base.

To define connectivity for a new Access knowledge base (AKB) for a local prototyping implementation

1. Confirm that the target environment has appropriate Microsoft Access licensing.

2. After launching HaleyAuthority, the HaleyAuthority dialog box displays. Under Create a new knowledge base using, click the New knowledge base file (*.akb) radio button, then click OK.

3. Name the knowledge base, for example TestCallBack, and click Save.

   HaleyAuthority creates its tables and initializes the knowledge base.

Migrating Rules Between Environments

You will want to migrate the knowledge base and runtime data from one environment to another environment; for example, from your development environment to your production environment.

In some cases you want to migrate knowledge bases and runtime data from more than one source environment to the target environment; for example, from individual developers' local prototyping environments to the common development environment.

It is recommended that you migrate a knowledge base and runtime data from one source environment to the target environment to minimize the chance that errors occur. This topic emphasizes the one-to-one migration, but provides requirements and guidelines for migrating data from more than one source environment.

Migrating rules data between environments involves the following tasks:

- "Migrating Knowledge Base Data Between Environments" on page 40
- "Migrating Rules Runtime Data Between Environments" on page 41

Migrating Knowledge Base Data Between Environments

You can migrate a knowledge base (concepts and relations) to a new knowledge base or to an existing datasource by using the Backup utility in HaleyAuthority.

You cannot use the Backup utility to migrate a knowledge base to a datasource with existing knowledge base tables. Thus two developers cannot both backup their local knowledge bases to the target environment. Only one may do so. Subsequent objects must be imported manually into the target knowledge base.

CAUTION: To better control backups, you may want to restrict the systems each user can access so that only certain users can do backups.

NOTE: You can also use the Backup utility to roll back a knowledge base back to a prior state.

To migrate data in a knowledge base to the target environment

1. Open the source knowledge base in HaleyAuthority.
2. Choose Tools > Backup, and choose one of the following, depending on your situation.
   - **Into new HaleyAuthority Knowledge Base.** Choose this option to make a local copy of the knowledge base as a new Access knowledge base.
   - **Into existing ODBC datasource.** Typically, choose this option to migrate the knowledge base data to a new target environment on a database server.
3. Save the knowledge base to the target environment:
   - If you are backing up to a new knowledge base, then specify that knowledge base file and location in the Specify HaleyAuthority Knowledge Base dialog box.
   - If you are backing up to an existing ODBC datasource, then select the target datasource in the Knowledge base selection dialog box.

     NOTE: You are disallowed from backing up to a datasource that has existing knowledge base tables, with or without data.

For subsequent developers to migrate their HaleyAuthority knowledge base objects to the target HaleyAuthority knowledge base, they must manually import those same Siebel objects to the target HaleyAuthority Knowledge base.

NOTE: Multiple developers are disallowed from concurrently importing objects into a knowledge base. A message displays to those who attempt to import objects while an import is in progress.

An alternate method to migrate knowledge bases from multiple sources is to manually import all of the objects (in serial sessions), instead of backing up the first knowledge base.

For information about importing objects, see "Using Siebel Object Importer" on page 47.

Migrating Rules Runtime Data Between Environments

The following topics describe the two primary methods for migrating runtime data (rule modules and statements) and other Object Importer metadata that is used at runtime from one environment to another.

- "Migrating Rules Runtime Data Using Export and Import Utilities" on page 41
- "Migrating Rules Runtime Data Using Siebel Application Deployment Manager" on page 42

Migrating Rules Runtime Data Using Export and Import Utilities

You can migrate runtime data (rules modules and statements) and other Object Importer metadata that is used at runtime to a target environment by using the Export and Import utilities in your Siebel application.

This strategy is intended for migrating small numbers of rules to a single target environment. For larger amounts of data, or for migrating rules and other Siebel data concurrently, or for migrating data to multiple servers, you typically use Siebel Application Deployment Manager (ADM). For more information, see "Migrating Rules Runtime Data Using Siebel Application Deployment Manager" on page 42.

Runtime data is exported to XML files from a source runtime datasource. The XML is imported into the target runtime datasource.

Observe the following requirements for migrating rule modules:

- If you must migrate rule modules from multiple source environments, then the name of each module must be unique within the entire collection of modules that is migrated to the target environment.

  CAUTION: Modules overwrite current data. A migrated module overwrites an earlier migrated module of the same name. Only the state of the original module in the target environment is preserved.

- The semantic role model on which imported rule modules are based must be consistent with the Siebel Repository for the target datasource. To ensure consistency, the repository for the source datasource must be identical to the repository for the target datasource.

To export runtime data

1. In your Siebel application, navigate to Administration - Business Rules > Rule Modules List
2. In the Rule Modules applet menu, choose one of the following:
   - **Export All Modules with Model**. If the semantic role models in the target (import) and source (export) environments differ, such as when no rule modules are yet deployed in the target environment, then you must export all rule modules from the source environment with the model.

     NOTE: You must not export a subset of rule modules with the model.

- **Export Modules Only**. If the semantic role model exists in the target (import) environment's runtime data and is identical to the semantic role model in the source (export) environment's runtime data, then there is no need to replace the target environment's runtime role model. You can export selected modules only.

An XML file with the exported rule modules is created. If applicable, it includes the semantic role model for the exported rules.

3 Opt to save the file.

4 Repeat Step 1 through Step 3 in each runtime environment from which you must export rule modules.

To import runtime data into the target datasource

1 In your Siebel application, navigate to Administration - Business Rules > Rule Modules List

2 In the Rule Modules applet menu choose Import Rules.

3 In the Import Rules dialog box, browse to select an XML file that was exported from the source runtime datasource.

4 Repeat Step 2 and Step 3 for each exported XML file from all source environments.

Import preserves the rule module state in the target environment if it is already deployed there. That is, if the existing rule module of the same name in the target environment has active state, then the imported rule module has active state.

If an imported rule module does not have a rule module of the same name already deployed in the target environment, then the imported rule module assumes the state it had when it was exported from the source environment. That is, if the rule module was in active state when it was exported, then it assumes active state when it is imported.

For information about activating a rule module, see "Configuring and Activating a Deployed Rules Module" on page 63.

Migrating Rules Runtime Data Using Siebel Application Deployment Manager

Application Deployment Manager (ADM) is the framework that automates the migration of enterprise customization data from one Siebel environment to another.

This topic provides the rules-specific details for using ADM to migrate rules-related data. For detailed information about using ADM in all contexts, see *Siebel Application Deployment Manager Guide*.

For migrating small numbers of rules, HaleyAuthority's Import and Export functions suffice. However, Import and Export are not practical for migrating all the components of a particular customization that includes rules and other Siebel data. Such customizations may include migrating rules, as well as objects such as business components, views, and lists of values (LOVs). To migrate such customizations efficiently, use ADM to create packages and deploy.

Business Rule is a predefined ADM data type that is intended for use in migrating business rules runtime data.

Please consider the prerequisites and principals when you use ADM to migrate rule modules:

- The source and target environments must have identical knowledge bases.
- It is assumed that the source and target environments for the migration are derived from a common environment. Exporting the semantic role model from the source to the target environment replaces the model in the target environment, so it is important that most, if not all, existing rule modules in the target environment also exist (modified or not) in the source environment. Thus, after migration, the model required in the target environment is essentially updated with no deficiencies for rule modules that existed in the target environment, but not in the source environment, prior to migration.

  If the semantic role model is replaced in the target environment during migration, any rule modules that existed in the target environment, but not in the source environment, may have their Inconsistent flag set to Y in the Rule Modules applet in the Administration - Business Rules screen. You must resolve any rule module's inconsistency with the new model in HaleyAuthority and re-deploy the rule module from HaleyAuthority. For more information, see "Configuring and Activating a Deployed Rules Module" on page 63.
- After migration, all modules have an Active status displayed in the Rule Modules applet in the Administration - Business Rules screen. If the status of a module should not be Active, deactivate the module. For more information, see "Reconfiguring and Deactivating Rule Modules" on page 65.

To migrate rule modules using ADM

1. In the Siebel application that points to the datasource to which you deployed your rules (the source environment), choose Site Map > Administration - Business Rules >
2. Choose Site Map > Application Deployment Manager > Deployment Projects. Create a new deployment project using the instructions in *Siebel Application Deployment Manager Guide*. All the fields that need to be filled in for the project are explained in the *Siebel Application Deployment Manager Guide*.

   Use the following guidelines to clarify the entries for some specific fields:

   - **Session Configurable.** Your entry in the Deployment Filter field, described in the following Step 3, determines which rule modules are exported in this project. To override this filter for a particular deployment session, check this field. To prevent a session-level override of the filter, leave this field unchecked.
   - **Target System** and **Target User Name.** No entries are necessary.
3. In the lower applet, follow the instructions in *Siebel Application Deployment Manager Guide* to create a deployment filter for selecting the subset of rule modules to export.

   Use the following guidelines to clarify the entries for some specific fields:

   - **Data Type Name.** Enter Business Rule.
   - **Deployment Filter.** The Business Rule ADM data type supports the following syntax to implement a field-level search specification on the Rule Module business component. You must enter the search specification manually.

```
[Field Name] operator 'Filter_Criterion' -model
```

The optional `-model` flag specifies to export the selected rule modules without exporting the source environment's semantic role model.

Use the following guidelines for setting the `-model` flag:

- Append `-model` if the semantic role model exists in the target environment's runtime data and is identical to the semantic role model in the source environment's runtime data. For this scenario, there is no need to replace the target environment's runtime role model, so setting `-model` increases the migration performance. For example, to export only rule modules whose Name field begins with *Pharma*, without the model, enter:

  ```
  [Name] like 'Pharma*' -model
  ```

- If the semantic role models in the target and source environments differ, such as when no rule modules are yet deployed in the target environment, then you must export all rule modules from the source environment with the model. To do so, leave the Deployment Filter field empty or enter:

  ```
  [*]
  ```

NOTE: You must not export a subset of rule modules with the model. That is, a filter specification like the following is not valid:

```
[Name] like 'Pharma*'
```

4 Use the instructions in *Siebel Application Deployment Manager Guide* to backup rule modules in the target environment, import the rule modules, activate, and restore the rule modules as you would do for any other ADM data type.

NOTE: During restore, any new rules that are inserted during import are deleted. To change this behavior, modify the ADM registry by changing the value of the DeleteNewObjects property to N. For information on the ADM registry, see *Siebel Application Deployment Manager Guide*.

5 Depending on the parameters you set in ADM, the state of imported modules (Active or Inactive) may not be the same state as the modules they update. Check the states of the updated modules, and activate or deactivate modules as needed.

Migrated modules are in Active state provided the Activate step of the ADM migration process is invoked. If Activate is not invoked, then modules' states prior to migration are preserved after migration.

For information on activating rule modules, see "Configuring and Activating a Deployed Rules Module" on page 63. For information on deactivating rule modules, see "Reconfiguring and Deactivating Rule Modules" on page 65.

Best Practices for Configuring and Maintaining Your Rules Development Environment

Table 2 provides best practices to follow when you configure and maintain your rules development environment.

Table 2. Best Practices for Configuring and Maintaining Your Rules Development Environment

Guideline	Explanation
Do not use a knowledge base to update more than one runtime database.	Your edited or added relations may not be consistent with the Siebel object model.
Do not delete a knowledge base.	Deleting a knowledge base deletes the knowledge base tables in the runtime database to which the knowledge base points.
Run only one knowledge base against a runtime database. That is, do not create a new knowledge base against a runtime database to which another knowledge base points.	When you import objects for the new knowledge base, the runtime data created for the original knowledge base is deleted, making the original knowledge base useless.
Backup runtime tables nightly using an SQL tool.	This is an extra precaution to protect against data corruption.
Never modify or delete any HaleyAuthority concepts, relations, or phrasing manually, especially objects created by the Object Importer.	A knowledge base is not supported if imported Siebel objects' concepts, relations, or phrasing are modified or deleted. See "About Technical Support for Rules" on page 193.

7 Creating and Deploying Rules

This chapter provides information on creating, testing, and deploying rules in HaleyAuthority. The chapter contains the following topics:

- "Using Siebel Object Importer" on page 47
- "Creating Rule Modules" on page 52
- "Using the HaleyAuthority Test Harness" on page 56
- "Deploying Rule Modules" on page 59
- "Using Object Importer and Deployment Log Files" on page 60
- "Best Practices for Creating and Deploying Rules" on page 60

Using Siebel Object Importer

You use the Siebel Object Importer to do two types of task:

- "Importing Siebel Objects" on page 47
- "Synchronizing the HaleyAuthority Knowledge Base with the Siebel Repository" on page 50

Importing Siebel Objects

In order to compose rules to act on Siebel objects, you must first:

- Generate a minimal set of HaleyAuthority objects that are required for implementing rules in Siebel applications. This import happens implicitly when you do your first import of Siebel objects for a new HaleyAuthority knowledge base.

  For detailed information, see "Siebel Concepts, Relations, Actions, Functions, and Predicates" on page 174.

- Import Siebel objects into HaleyAuthority. You expicitly choose to import the Siebel objects on which your rules are based. See "Using Siebel Object Importer" on page 47.

When you import specific Siebel business objects, business components, and fields on which to build your rules, Siebel Object Importer automatically creates HaleyAuthority concepts and relations to represent those objects.

For detailed information on the mapping between Siebel Objects and HaleyAuthority concepts and relations, see "Siebel Object Model Mapping to the HaleyAuthority Semantic Role Model" on page 170.

Table 3 provides the fundamental relations and their phrasings that are automatically created in HaleyAuthority by Object Importer. Names of automatically-created relations are of the form:

siebelparentobject_siebelchildobject

The object names are rendered in lower case with no spaces and are connected with an underscore.

Table 3. Relationships Automatically Created in HaleyAuthority by Object Importer

Siebel Objects	Name of Relation	Phrasing of Relation
Parent business component and child business component are both imported. **NOTE:** When a child business component is imported, its parent business component is automatically imported.	*parentbusinesscomponent_childbusinesscomponent*	a *parent business component* has a *child business component* and equivalents
Business component and a single-valued child field are imported.	*businesscomponent_field*	a *business component* has a *field* and equivalents
Business component and a multi value child field are imported.	For details of multi value field imports, see "Siebel Object Model Mapping to the HaleyAuthority Semantic Role Model" on page 170	

Note the following Object Importer restrictions and limitations:

- You can only import one Business Object at a time.
- The following fields cannot be imported:
  - Non-UTC date/time fields; that is, fields of Type DTYPE_DATETIME
  - Calculated fields without Type
  - Fields of Type DTYPE_CLOB
- You must not manually modify concepts and relations that are built among Siebel objects in HaleyAuthority. Such changes must result only if the relations change in the Siebel Repository, and you then make those changes by using Object Importer to synchronize the HaleyAuthority knowledge base with the Siebel Repository.

Once a particular Siebel object is imported, it can be used by anyone with access to the HaleyAuthority authoring environment to build other rules.

Before importing Siebel objects into HaleyAuthority, see Using Object Importer and Deployment Log Files for information on enabling the Object Importer log file.

To import Siebel objects into HaleyAuthority

1. Launch HaleyAuthority in either of the following ways:
   - Choose Start > Program Files > *Program Group that contains Siebel Tools* > Siebel Business Rules Developer.

- Choose Start > Program Files > Haley Systems > HaleyAuthority Enterprise.

2 In HaleyAuthority, choose File > Import > Siebel Object.

Siebel Object Importer Wizard launches.

3 From the Siebel Object Importer Wizard Welcome screen, click Next.

4 In the Login screen, enter database connection parameters, then click Next. For each of the Master Repository database and the runtime datasource:

- Specify (or browse to) the Siebel application cfg file (for example, uagent.cfg or siebel.cfg) which points to the applicable database. It is typically located in the \BIN*LANG* directory in your Siebel server root directory (or Siebel client root directory for a local database), where *LANG* is the language code (for example, ENU).
- Enter the login credentials to the applicable database.
- Choose the datasource - ServerDataSrc, GatewayDataSrc, Sample, or Local.
- For the runtime data connection, also choose Server or Local mode.

5 In the Pick Task screen, click the Import Siebel objects radio button, then click Next.

6 From the Pick Business Object screen, select one business object from the picklist, then click Next.

NOTE: If you import a business component or field that has been imported previously, then Object Importer does not create duplicate concepts

7 The Pick Business Components screen lists the hierarchy of business components of the business object you chose in Step 6. Click the checkbox of each business component of this business object that you want to import. Click expansion icons to see and select child business components. Click Next.

NOTE: If you expand a business component to select a child business component, then its parent business component is automatically selected.

NOTE: You can import different fields for a business component by importing the business component more than once, adding new fields in subsequent imports.

8 The Pick Business Component Fields screen lists the fields of the first business component that you chose in Step 7. In the Available Fields list, click each field you want to import. Click the right arrow to move the selected fields to the Selected Fields list or click the double right arrow to move all of the available fields. Click Next.

The fields for the next business component you chose in Step 7 display in the Available Fields list. Repeat this step to select fields for each business component you chose in Step 7.

9 The Finish screen displays the hierarchy of business object, business components, and fields to be imported. Confirm to import this hierarchy by clicking Finish.

10 To import more objects, click Yes, else click No.

The Output screen displays a log of the concepts, relations, and dictionary objects that are automatically created in HaleyAuthority.

NOTE: For multi value fields (MVFs), concepts are created for the target business component of the MVF and for its primary.

For overview information about Siebel Object Importer, see "About Siebel Object Importer" on page 29.

Synchronizing the HaleyAuthority Knowledge Base with the Siebel Repository

The HaleyAuthority semantic role model (representation of Siebel objects as HaleyAuthority concepts and relations) must be kept consistent with the Siebel object model. Changes to the Siebel object model, such as the following, require that you synchronize the HaleyAuthority knowledge base with the Siebel Repository:

- In general, any Siebel object that is used in any rule, whose definition is changed in the Siebel Repository, must be synchronized.
- An imported field's Type changes in the Siebel Repository, such as from DTYPE_TEXT to DTYPE_BOOL.
- An imported field is inactivated in the Siebel Repository. A new field is created to access the same underlying table column.
- Imported objects are affected by a schema change in the Siebel database, such as may occur after an upgrade.

You can synchronize the HaleyAuthority knowledge base with the Siebel Repository implicitly or explicitly.

After synchronizing, rule statements that were previously understood by HaleyAuthority may become draft statements, or not understood. Draft statements display with a question mark in the HaleyAuthority explorer. You must edit these statements to make them understood before re-deploying the rule modules that contain them.

Synchronizing the HaleyAuthority Knowledge Base Implicitly

You can synchronize HaleyAuthority concepts with individual Siebel objects by re-importing the Siebel objects into HaleyAuthority.

For example, suppose the Type of the Marketing Sub Plans business component's Actual Expenses field was changed from DTYPE_NUMBER to DTYPE_CURRENCY in the Siebel Repository, and you previously imported this business component and field into your HaleyAuthority knowledge base. Use Object Importer to re-import the Actual Expenses field.

For detailed information about importing Siebel objects into HaleyAuthority, see "Using Siebel Object Importer" on page 47.

The following changes are made in the HaleyAuthority knowledge base and are visible in HaleyAuthority:

- The concept actual expenses is deleted in the HaleyAuthority concepts tree under quantity: number: decimal number: real number and under value: number: decimal number: real number.
- A new concept actual expenses is created in the HaleyAuthority concepts tree under entity: currency value.

- The relation marketingsubplans_actualexpenses is deleted.
- A new relation marketingsubplans_actualexpenses is created.

To see added relations, choose View > Tabs > Full View and look in the Relations and Procedures hierarchy.

Synchronizing the HaleyAuthority Knowledge Base Explicitly

You can synchronize the HaleyAuthority knowledge base with the Siebel Repository:

- For all imported Siebel objects
- For selected imported Siebel objects

Deletion or modification in the Siebel Master Repository of an imported Siebel object results in the object being deleted in the HaleyAuthority knowledge base and, if applicable, a new object is created.

To explicitly synchronize the HaleyAuthority knowledge base with the Siebel Repository

1. In HaleyAuthority, choose File > Import > Siebel Object.

   Siebel Object Importer Wizard launches.

2. From the Siebel Object Importer Wizard Welcome screen, click Next.
3. In the Login screen, enter database connection parameters, then click Next. For each of the Master Repository database and the runtime datasource:
   - Specify (or browse to) the Siebel application cfg file (for example, uagent.cfg or siebel.cfg) which points to the applicable database. It is typically located in the \BIN\LANG directory in your Siebel server root directory (or Siebel client root directory for a local database), where *LANG* is the language code (for example, ENU).
   - Enter the login credentials to the applicable database.
   - Choose the datasource - Server, Sample, or Local.
   - For the runtime data connection, also choose Server or Local mode.
4. In the Pick Task screen, do either of the following, then click Next.
   - Click the Synchronize selected Siebel objects radio button to synchronize a subset of your imported Siebel objects.
   - Click the Synchronize entire model radio button to synchronize all of your imported Siebel objects.
5. If you chose to synchronize selected objects in Step 4, select one or more business objects from the picklist on the Pick Business Objects screen, then click Next.
6. On the Finish screen, click Finish.
7. To perform another import or synchronization task on the same database, click Yes, else click No.

   The Output screen displays a log of the concepts and relations that are updated.

Creating Rule Modules

This topic includes basic instruction on creating rules and rule modules. This topic is not intended as primary documentation for this instruction. For detailed information about creating rule modules, see the Haley documentation that is included in HaleyAuthority through the Help menu.

For examples of creating rule statements and rule modules, see:

- "Scenario for Using Rules to Validate Data at Runtime" on page 78
- "Scenario for Using Rules in a Siebel Workflow to Do Batch Processing" on page 104
- "Scenario for Using Rules to Provide Dynamic Navigation in a Siebel Task" on page 128
- "Scenario for Validating Data Using Script to Invoke Rules" on page 94

Once Siebel objects are imported, you can create rule modules to implement business logic. A rule module is a group of rules statements that you want to apply to imported objects.

HaleyAuthority allows you to define rules as English statements that:

- Identify business outcomes and specify actions to take.
- Compose if-then-else constructs via definitions of rules statements:
  - **if conditions.** The conclusion that is associated with a set of if conditions is true if at least one of the if statements is true (or).
  - **only if conditions.** The conclusion that is associated with a set of only if conditions is true if all of the only if statements are true (and).
  - **unless conditions.** The conclusion that is associated with a set of unless conditions is false if at least one of the unless statements is true (*not*).
  - In the case of more than one of the condition types being applied to the same conclusion, the order of precedence is unless, then only if, then if.

To create a rule module, you need to do the following tasks:

- Add the rule module.
- Add statements to the rule module.
- You may add submodules to a rule module and statements to the submodules, in order to provide more organization for your rules.

To add modules and submodules

1. In HaleyAuthority, navigate to the Modules and Statements View. If the Modules and Statements tab is not available, choose View > Tabs > Modules and Statements.
2. Select Modules and Statements in the explorer, then choose Object > Add a module.

3. Enter a name for the module.

   CAUTION: Do not include any special characters in the module name, such as * (asterisk), ? (question mark), / (slash), ! (exclamation), @ (at), or quotes. Also, the entire module name cannot be a Windows reserved word, although you may have a reserved word as a substring in the name. For example, *Incoming SRs* is acceptable, but *Com* alone is not acceptable.

4. To add a submodule to a module, select the module and choose Object > Add a submodule. Name the submodule.

You can add more modules and submodules as needed.

To add a statement to a module or submodule

1. In the Modules and Statements view, select the module or submodule, then choose Object >Add a statement.
2. In the Edit statement dialog, type in a statement. Alternatively, you can pick from the Words list at any time to enter allowed words at any place in the sentence. The Words list provides the only words that can be understood by HaleyAuthority at the current place in the statement.

   Use double quotes to differentiate a string value (such as *my dog*) from words that are critical to the structure of the sentence. Any string value can be used in a place where a string value is allowed structurally.

3. Edit the syntax until the entire statement is bold and the *not understood* message changes to *understood*, then click OK. Alternatively, you can click OK, and return later to correct the statement.

   NOTE: A rule module that contains a draft statement (a statement that is not understood by HaleyAuthority) cannot be deployed.

Writing Rules that Involve Parent and Child Business Components

When you write rules, you must keep in mind the intent of your rules and the context in which they are executed. Possible errors that can occur include:

- An intended business process is enforced sometimes, but not all the time, because a rule is not executed in all applicable contexts.
- One or more rules create a logical paradox; that is, one or more rules incorrectly prevent the satisfaction of the conditions for a rule, thus preventing it from executing at appropriate times.
- A rule module may execute in more contexts than intended, such as for all contexts of a business component instead of in a restricted context.

For example, the Expense item business component is a child business component of the Expense business component. Your business processes require that certain conditions be true of expense items before their parent expense is valid.

The following rules are examples of rules that need improvement:

Table 4. Rules Needing Improvement

Rule	Weakness	Improvement
`if an expense has any expense item which does not have a description then invalidate the expense with "All expense items should have a description."` is called in a PreWriteRecord runtime event on the Expense business component.	The rule prevents an expense record from being written if it has an expense item with no description, but it does not prevent an expense item from having no description. A user may create an expense item without a description, then never return to the parent expense item on which the rule is called.	Write the rule on a PreWriteRecord event for the expense item business component, instead of on the expense business component. Then every expense item will be required to have a description.
`if an expense has no expense item then invalidate the expense with "This expense does not have any line items."` is called in a PreWriteRecord runtime event on the Expense business component.	This rule disallows saving an expense record unless it has an expense item. However, you cannot add an expense item to an expense while the focus is the Expense business component. You add an expense item to an existing expense in the Line Items applet, whose business component is Expense Item. Thus you can never add an expense item to an expense, because you cannot create an expense without an expense item.	This is not an issue because an expense report has a zero total if it has no contributing expense items. Instead of a hard invalidation, you can instead provide a reminder to the user to add expense items, such as adding the following rule to a WriteRecord event on the expense business component: `if an expense has no expense item then raise an error as "Remember to add expense items to this expense report."`

Typically, the best practice is to attach a rule to the business component to which the rule most directly applies, and to appropriately restrict the context in which the rule should apply.

Many business components can be imported in either of two ways:

- As the primary business component of the business object that you are importing
- As a child business component within the business object that you are importing

Independent of which way you import a business component, any rule written on the business component executes in all contexts, not just when the business component is playing one or the other of the parent or child roles. For example, a rule defined on the Action business component, imported as a child business component within the Opportunity business object, also executes in the Activity screen where Action is the primary business component.

To see an example of choosing the appropriate business component on which to write rules, see "Scenario for Using Rules to Validate Data at Runtime" on page 78.

Using Some Standard Rule Constructs

This topic illustrates common errors and standard rule constructs for some common scenarios.

In some of these examples, a rule statement may be syntactically correct, but semantically incorrect. That is, HaleyAuthority may indicate that a rule statement is understood, but the rule statement may not function as you intend it to.

- **Setting a field value.** You intend a rule to set a value in the First Name field for a contact. HaleyAuthority understands the following rule statement:

  ```
  set the first name of the contact to "Ricardo"
  ```

  However, the field is not set by the rule because the correct semantic form for setting a field value is to quote the field's name, capitalize the field name as it is in the Siebel repository, and to not precede the field name with an article. The correct statement is:

  ```
  set “First Name” of the contact to "Ricardo"
  ```

- **Evaluating currency fields.** You intend a rule to raise an error if the amount on an expense item exceeds $200. HaleyAuthority understands the following rule statement:

  ```
  if an expense item has an amount that is more than 700 in "USD" then raise an error
  as "This expense item exceeds the maximum limit."
  ```

  However, currency field comparisons, like the comparison with the Amount currency field here requires all three of the following components in the comparison amount: number of units, currency type, and the exchange date on which to compare. The correct statement is:

  ```
  if an expense item has an amount that is more than 700 in "USD" on an exchange date
  today then raise an error as "This expense item exceeds the maximum limit."
  ```

- **Evaluating a multi-value field.** You intend a rule to raise an error if an opportunity has a revenue of more than $10,000. An opportunity can have more than one revenue. The following statement seems to be reasonable, but is not understood by HaleyAuthority:

  ```
  if an opportunity has a revenue that is more than 10000 in "USD" on an exchange date
  today then raise an error as "Revenue is more than $10000."
  ```

  For this example, you imported the Opportunity business component and its Revenue field. The Revenue field is of type currency and is a multi-value field. When you import a multi-value field, the Object Importer also seamlessly imports the business component on which the multi-value link is based (Revenue business component) and the target field on that business component (Revenue field on the Revenue business component).

  The rule is not understood because it is comparing the Revenue business component against an amount, which is an invalid comparison. HaleyAuthority understands the statement up to the comparison, as indicated by bold and non-bolded text.

  The following statement correctly compares the Revenue field of the Revenue business component against the amount:

```
if an opportunity has a revenue and the revenue's revenue is more than 10000 in "USD"
on an exchange date today then raise an error as "Revenue is more than $10000."
```

For information on the HaleyAuthority concepts that are created automatically when you import a multi-value field, see "How the Multi-Value Revenue Field is Imported" on page 148, which is a topic within the example presented as "Scenario for Using Rules to Provide Dynamic Navigation in a Siebel Task" on page 128.

Writing Rules Using Language Independent Code (LIC)

By default you write rule statements that use language specific values. Alternatively, you can write rule statements that reference Siebel string values that are language independent.

Consider the following statement:

```
if an account's name contains "ABC Company" then set "Account Status" of the account
to "MOST_PRECIOUS"
```

If you set your implementation to interpret LIC, then the Siebel string value MOST_PRECIOUS is interpreted depending on the language that is implemented in your Siebel implementation. For English-American, it is translated to *gold*.

If your implementation is not set to interpret LIC, then the literal *MOST_PRECIOUS* is written to the Account Status field, independent of the language you are implementing.

NOTE: HaleyAuthority only supports rules authoring in English. However, actions performed in your Siebel application can leverage LIC.

To set your implementation to interpret LIC when executing rules

- In the cfg file for the application, for example uagent.cfg for Siebel Call Center, set the following parameter:
  - In server mode, set

    ```
    BizRuleUseLIC = TRUE
    ```

  - For the Developer client or Mobile Web Client, create the following section:

    ```
    [BizRule]

    BizRuleUseLIC = TRUE
    ```

Using the HaleyAuthority Test Harness

You can use the test harness in HaleyAuthority to test your rules in a simulation environment before deploying the rule module. This simulation does not involve the runtime database or Siebel application environment.

CAUTION: The test harness does not support testing rule statements that involve currency.

Testing your rules in HaleyAuthority includes the following tasks:

- Create a test case.

- Create example instances of the entities that the rule module processes.
- Run the test case.
- Examine the test results.
- Modify the data and rerun the test case. (optional)

For this topic, assume your knowledge base has the following rules module and the concepts and relations that are implied:

```
if a service request does not have a priority then set “Priority” of the service
request to “3-Medium”

if the priority of a service request is “1-ASAP” then set “Recommendation” of the
service request to “Escalate”
```

To create a test case

1. In HaleyAuthority, choose View > Tabs > Tests & Cases.

   The Tests & Cases window displays.

2. Right click the Test Cases folder and choose Add a test case.
3. In the Edit Test Case dialog box, enter a name for the test case in the Description field; for example, SR Priority. Click OK.

Create instances of concepts that serve as simulated data on which rules act. You must create instances for each concept that the rules reference.

To create example instances of concepts

1. Choose View > Tabs > Concepts.

   The Concepts window displays.

2. Locate and right click a concept for which to create an instance, then choose Add > an instance or example. For this example, select the service request entity.
3. In the instance dialog box, make the following entries, then click OK:
   - A label for the instance; for example, test_sr_1.
   - Click the *This is an example for testing purposes only* checkbox.

   The Instances folder under service request now contains the test_sr_1 instance.

4. Right click on test_sr and choose Properties, then click the Facts tab.
5. Add one or more facts to your instance. Click the New icon and select a phrasing for your fact; for example, *test_sr_1 has a priority*. For the priority value, enter *1-ASAP*, then click OK.

   NOTE: Only add a fact to an instance if the associated property has a value. For example, to test using a service request instance that has no priority value, do not add a *has a priority* fact to the instance.

6. Associate applicable test cases with the instance. For example, right click the test_sr_1 instance and choose Test cases, click the SR Priority checkbox, then click OK.

   NOTE: Alternatively, you can add all example instances to a test case by right clicking the test case in the Tests & Cases window and choosing Examples.

This example is intended to act on one service request at a time, so associate only one service request at a time with this test case.

To run a test case

1. Choose View > Tabs > Tests & Cases.

   The Tests & Cases window displays.

2. Locate and right click the applicable test case (SR Priority in this example) and choose Test case.
3. In the HaleyAuthority Enterprise dialog box, click Yes to deploy (or redeploy) your logic.
4. In the Deploy Configuration dialog box, make the following entries, then click OK.
   - Accept the default of the Deploy checkboxes being checked for all modules.

     NOTE: You may uncheck other named modules that are not part of your test for performance purposes.

   - Enter or browse to choose a directory for the deployed module; for example, D:\temp.

     NOTE: This directory has no persisting value, and does not affect the eventual runtime database to which the module is deployed.

   The test case runs.

The test case provides two types of output:

- The Output window provides a log of the statements that were executed. These include unconditional action statements, conditional statements for which the if clause is true, and applicability statements that are true.

  NOTE: A test case runs against all rule modules in the knowledge base for which the test case's example instances are applicable. You cannot restrict a test case to run against a specific rule module.

  NOTE: Siebel-specific functions (imported on your initial import with Object Importer) result in a *Loaded undefined function 'function name'* line. However, the output indicates that the simulated test executes those functions.

- An entry of the form *<test case name> on <date and time>* is added to the Test Results folder under *test case name* folder. Expand this folder to see the statements that were executed. For this example, one statement displays:

  ```
  1 if the priority of a service request is "1-ASAP" then set "Recommendation" of the
  service request to "Escalate"
  ```

  None of the other statements' conditions are satisfied.

You can test different aspects of your rule module by doing one or both of the following:

- Modify existing example instances and rerun the test.
- Create new example instances, disassociate the existing instances from the test case, and associate the new instances with the case. For this example, only associate one service request instance at a time with the test case.

  For this example, you could create the following service request example instances and test them one at a time after disassociating the test_sr_1 instance from the SR Priority test case.

For an example that illustrates use of the test harness, see "Scenario for Using Rules to Validate Data at Runtime" on page 78.

Deploying Rule Modules

For any rule module to be available for implementation in the Siebel application, the rule module must first be deployed from HaleyAuthority. Once deployed, the rule module is visible and available in the Rule Modules List view of the Administration - Business Rules screen.

In order to deploy a rule module, the module must contain at least one statement that is understood.

The Deployment plug-in points by default to the runtime datasource that was specified in the Object Importer session you ran for the HaleyAuthority knowledge base.

NOTE: The list of rule modules that are available to deploy does not include rule modules that contain draft statements (statements that are not understood by HaleyAuthority).You cannot bring up the Deployment plug-in for a HaleyAuthority knowledge base without running Object Importer at least once.

Before deploying rule modules, see Using Object Importer and Deployment Log Files for information on enabling the Deployment log file.

To deploy a rule module in HaleyAuthority

1. Choose Tools > Siebel Deployment.
2. In the Siebel Deployment dialog box, in the List of Modules, select one or more modules to deploy.
3. Use the following guideline for checking or not checking the Update model checkbox:
   - It is recommended that you check the Update model checkbox if you have changed your HaleyAuthority semantic role model (the entire set of concepts and their relations) in any way since the last deployment of a rule module; for example, by importing new fields or doing a synchronization with the Siebel Repository. Choosing this option loads the updated compiled semantic role model from HaleyAuthority into the runtime database.
   - If the semantic role model has not been updated since the last rule module deployment, then you can uncheck the Update model checkbox. However, for a little performance overhead, it is likely worth leaving Update model checked whenever you deploy.
4. Click OK.

When a rule module is deployed for the first time, its status is Inactive in your Siebel application. To configure and activate a rule module, see "Configuring and Activating a Deployed Rules Module" on page 63.

Using Object Importer and Deployment Log Files

Additional Siebel-side logging can be enabled for the Object Importer and Deployment plug-ins. Set the environment variable BZR_PLUGINS_LOG to value 1 on the machine on which you are running HaleyAuthority.

Using the Object Importer and Deployment plug-ins after you set the BZR_PLUGINS_LOG environment variable generates the following log files, respectively, in your Haley installation folder:

- objectimporter_plugin.log
- deployment_plugin.log

You can use the information in these log files to troubleshoot any problems you encounter when using the Object Importer or Deployment plug-ins.

Best Practices for Creating and Deploying Rules

Table 5 provides best practices to follow when you create rules and rule modules.

Table 5. Best Practices for Creating Rules

Guideline	Explanation
Do not edit or add relations between the concepts that represent imported Siebel objects in the Concepts tree.	Your edited or added relations may not be consistent with the Siebel object model.
When a rule operates on a business component, it is not view-aware, and so it is not aware of the search specification on the business component in the view.	

Table 5. Best Practices for Creating Rules

Guideline	Explanation

To restrict applicability to the current record, applicability statements must have context. That is, the leading conclusion should use *a* or *an* to reference fields, and the applicability statements must use *the* to reference the fields.	If a field in an applicability statement uses *a* or *an* to reference a field, then the rule engine may look for satisfaction of the statement from another record.
	
	The following applicability construct applies the applicability statement to the current record only:
	
	`set "Description" of an opportunity to “Active”`
	
	`Applicability:`
	
	`if: the status of the opportunity is "Accepted"`

Name rule modules to reflect the context in which they are implemented.	There may be many modules in your implementation. Structured naming makes identification more efficient. For example, if a rule module is called by a business component runtime event, you could use the following construct for the module name: *business object_business component_event* A rule module that is called by the PreWriteRecord event for the Account business component of the Account business object is then named: Account_Account_PreWriteRecord
In a *set* statement, put double quotes around the name of the field whose value is set.	For example, in the following statement, the Description field is set: `if an account does not have a contact then set “Description” of the account to “Needs a contact.”`

Table 5. Best Practices for Creating Rules

Guideline	Explanation
When writing a rule that makes a comparison to a quoted string, be careful to duplicate the quoted string exactly.	String comparisons in rules are case-sensitive and space-sensitive. **NOTE:** HaleyAuthority's validation of a rule (statement that the rule is understood) does not include any verification of quoted strings.
Before running a test case, you must add every example instance that is referenced in the test case to the test case.	If some example instances are not added to a test case, then you can get inconsistent test results. For example, assume the following: ■ Example_expense is an example expense. ■ Example_expense_item is an expense item. ■ *Example_expense has example_expense_item* is a fact. ■ Example_expense is added as an example to sample_test_case, but example_expense_item is not added. Your test results may be inconsistent because some statements recognize that example_expense_item exists, while others do not.

8 Configuring and Activating Rule Modules

This chapter provides information on the Business Rule Service business service and instruction on configuring and activating rule modules. The chapter contains the following topics:

Configuring and Activating a Deployed Rules Module

To activate a rules module that you deployed from HaleyAuthority, you must:

- Confirm the following prerequisites:
  - The rules module exists and has been deployed from HaleyAuthority. See "Deploying Rule Modules" on page 59.
  - The Siebel client is logged in to the same datasource that you specified as the runtime datasource to use with Object Importer and to which you deployed rule modules from HaleyAuthority.
- Configure and activate the rules module in the Siebel client's Administration - Business Rules screen.
- Identify the implementation strategy you plan to use to execute the rule module and configure accordingly. For information about various implementation strategies, see "About the Business Rule Service Business Service" on page 66 and its subtopics.

To configure and activate a rules module

1. From the site map in your Siebel application, choose Administration - Business Rules > Rule Modules List.
2. In the Rule Modules List view, in the Rule Modules applet, locate the rule module that you want to configure.

   If the module is not listed, it may not yet be deployed from HaleyAuthority or you may be logged in to a different datasource than you ran against when you configured the module in HaleyAuthority.
3. In the Business Object field, pick the business object to which this rules module applies. This is the same business object from which you imported objects for this module.

4 If the Inconsistent (read-only) field has a value of Y, then you must abort this procedure, resolve the rule module's inconsistency in HaleyAuthority, and re-deploy the rule module from HaleyAuthority.

The Inconsistent flag indicates that this rule module is not consistent with the currently deployed semantic role model. For causes and remedies for this state, see "Troubleshooting Rules Configuration and Activation" on page 193.

5 The value of the Data Assertion Mode field determines whether one input data set to the Business Rule Service business service is asserted (loaded into memory) at a time, or all input data sets are asserted at once. An input data set here is a unit of transaction data consisting of the top-level business component and its child business components and fields, as defined by the BusCompList input property set. Pick one of the following values:

- **For All.** Choose this option to assert all input data sets at once. This is the default choice.
- **For Each.** Choose this option to assert one input data set at a time. Because only one input data set is in working memory at any time, this option minimizes the footprint of the working memory of the rules engine.

NOTE: For each particular collection of data sets that are being processed, performance may improve by choosing For Each instead of For All at a threshold number of data sets in the collection. At or above the threshold, performance improves. Because this threshold can differ depending upon the actual data, you may want to test for the threshold number in your development or test environment before deploying to your production environment.

6 In the detail Rule Module Relations applet, add a new rule module relation for each relation in the rule module that you are deploying. For each rule module relation, click the Business Component field, and pick a listed relation.

- Your rule module relations list must include the unary relation (the relation with no Parent Business Component entry) for each top level business component in the rule module
- Your rule module relations list must include all parent-child business component relations for all the business components that are used in the rule module.
- You can provide an additional search specification on the business components for each relation record you create.

7 Step off of the current rule module relation record to save it.

8 In the Rule Modules applet, with the appropriate rule module selected, click the Activate button.

After you activate the rule module, you cannot modify the rule module relations.

NOTE: The rule module and its associated relations are cached at runtime.

For information on rule module loading and caching, see "About Rule Modules Invocation: Loading, Using, and Caching" on page 36.

Reconfiguring and Deactivating Rule Modules

When a rule module is deployed for the first time, its state is Inactive. While the rule module is in Inactive state, you can configure the rule module's business object and assert mode, add rule module relations to the module, and activate it.

When a rule module is activated, its state changes to Active. While the rule module is in Active state, you cannot reconfigure its properties.

As related to runtime, when an active rule module is invoked for the first time, it is loaded into the memory and cached for future use.

To allow the properties of an active rule module to be reconfigured or to permanently deactivate the rule module, the Rule Modules List applet menu provides two options:

- Reconfigure Module
- Deactivate Module

Using the Reconfigure Module Option

When you choose Menu > Reconfigure Module in the Rule Modules List applet, the state of an active rule module changes to Published. While in the Published state, you can modify the rule module's properties and add new relations to it.

When a rule module is being reconfigured or when it is in Published state, the cached version of the module continues to run, assuming the module has been invoked at least once while in Active state.

After the configuration is done and you redeploy the rule module, reactivate the rule module by selecting it in the Rule Modules applet and clicking the Activate button. At this time the existing version of the module from the runtime cache is excised. Any new invocation of the rule module causes the new configuration of the module to be reloaded.

The advantage of having the Published state is that the rule module can continue to run with the cached version, thus reducing its downtime.

Using the Deactivate Module Option

You can deactivate a rule module that is in Active state by choosing Menu > Deactivate Module in the Rule Modules List applet. When a rule module is deactivated, its state changes to Inactive, and the cached version of the rule module is immediately excised. The rule module is completely unavailable to be run.

You can reactivate a deactivated rule module by selecting the rule module in the Rule Modules applet and clicking the Activate button.

You use the Deactivate Module option to stop running a rule module immediately for purposes of reconfiguring and reactivating the rule module or to stop running the rule module permanently.

About the Business Rule Service Business Service

The Business Rule Service business service is prepackaged in your Siebel Repository. Irrespective of the implementation strategy you choose, Business Rule Service is the underlying service that provides the interface to the Siebel rule engine.

The implementation strategies that can be configured to invoke the Business Rule Service business service are:

For information on the implementation strategies that can be configured to invoke the Business Rule Service business service, see:

- Chapter 9, “Integrating Rules with Siebel Runtime Events”
- Chapter 10, “Implementing Rules in Scripts”
- Chapter 11, “Implementing Rules in Siebel Workflows”
- Chapter 12, “Implementing Rules in Siebel Task-Based UI”

About the Input Property Set to Business Rule Service

RunRules is the only method invoked by the Business Rule Service business service.

Business Rule Service requires an input property set that consists of the input properties in Table 6.

Table 6. Business Rule Service Input Property Set Structure

Input Property	Data Type	Description
BusCompList	Hierarchy	This input property identifies and provides the hierarchical structure and data of the business components and fields that are required for the rule module that is run by the RunRules method. For each record on which to act, an element in this list includes: ■ The name of the top-level business component in the business object hierarchy on which this rules module acts ■ The Row_ID for the specific record of the top-level business component ■ If applicable, child elements that include names of required child and further descendent business components and field values that are required to evaluate the rule
GetMoreData	String	The default value of Y indicates that the *BusCompList* input property set needs only to specify the top-level business components. All descendent business components and fields that have been imported are automatically included in *BusCompList* implicitly. In addition, specific input field values are also included implicitly. A value of N indicates that all descendent business components, fields, and field values must be provided explicitly in the *BusCompList* property set. A value of Y minimizes the amount of script required to define the BusCompList input property. However, all of the top-level business components' descendent business components and fields that have been imported into the knowledge base are included in the input property set, instead of the minimal set required for the rule module to execute. Thus performance is not optimal. A value of N optimizes performance by providing only the minimal set of descendent business components and fields for the BusCompList input property that is required by the rule module to execute. However, you must provide the structure and data for this hierarchy explicitly, such as with script or as XML.

Table 6. Business Rule Service Input Property Set Structure

Input Property	Data Type	Description
PerformAction	String	The default value of this parameter is Y, indicating that all actions specified by the rule module, such as set-field-value or set-profile-attribute, are to be executed. A value of N indicates to not execute any actions. Typically, then, the caller of Business Rule Service processes the output property set, which contains the results of executing rules.
RuleModuleName	String	The names of the rule modules to invoke. Multiple rule module names must be separated by commas. Each rule module must first be deployed from HaleyAuthority, configured, and activated in the Siebel application.

The input property set for Business Rule Service is generated differently in different contexts.

- **Runtime event.** When you configure a runtime event to invoke rule modules, the Siebel runtime framework implicitly generates the input property set based on the values you enter in the Administration - Runtime Events screen when you configure the runtime event. You cannot control the structure of the BusCompList input property or the values of the other input properties.

  For information on configuring a rules module to run in a runtime event, see Chapter 9, "Integrating Rules with Siebel Runtime Events.".

  The input property set that is implicitly provided to Business Rule Service consists of:

  - BusCompList includes the top-level business components you specified in Administration - Runtime Events and all descendent business components and fields that have been imported into HaleyAuthority. In addition, field values in the current record are provided.
  - GetMoreData = Y, to allow for implicitly building the entire BusCompList property set, given only its top-level business components.
  - PerformAction = Y.
  - RuleModuleName is one or more names of rule modules that you specify in the Administration - Runtime Events screen.

- **Script, Task UI step, or workflow step.** The RunRules input property set must be provided explicitly, such as in script or in an XML file. Unlike the static structure of the input property set that is set implicitly for a runtime event, the actions that your rules execute and performance considerations determine your settings for GetMoreData, PerformAction, and the structure for BusCompList.

  Instruction for generating the input property set are provided in the following examples:

  - "Scenario for Using Rules in a Siebel Workflow to Do Batch Processing" on page 104
  - "Scenario for Using Rules to Provide Dynamic Navigation in a Siebel Task" on page 128
  - "Scenario for Validating Data Using Script to Invoke Rules" on page 94

To see the complete schema for the input property set, see "Input Property Set Schema for Business Rule Service" on page 185.

For requirements for setting the GetMoreData and PerformAction input properties, see "Setting the GetMoreData Input Property" on page 69 and "Setting the PerformAction Input Property" on page 70.

Setting the GetMoreData Input Property

Business Rule Service must be provided with data that it in turn asserts to the rule engine. Without data, rules do not execute.

Business Rule Service is provided data in one of the following ways:

- If GetMoreData = Y, then Business Rule Service gets data from an existing database record that is identified by the current business component rowid.
- If GetMoreData = N, then data is provided to Business Rule Service by its caller, typically script. That is, the caller must construct the data that is provided to Business Rule Service.

The following guidelines specify when setting GetMoreData=Y is allowed and when it is mandatory or highly recommended to set GetMoreData=N.

- **Direct invocation by a runtime event.** A runtime event invokes Business Rule Service directly when you declaratively configure the runtime event in the Administration - Runtime Events screen. When Business Rule Service is invoked directly by a runtime event, GetMoreData = Y by default and cannot be modified.

  The Siebel runtime event functionality controls the provision of data to Business Rule Service. Data is provided from the current record in the business component as needed. In the case of a *pre* event, such as PreWriteRecord, new data that is not yet written to the database is also provided as needed and where it is appropriate. For example, if a field value is changed in a PreWriteRecord, and the new field value is tested when a rule is executed, the rule evaluates the new, unwritten value instead of the existing value in the database.

  This allows a *pre* event to execute on an existing record that is modified or on a new record.

- **Indirect invocation by a runtime event.** A runtime event invokes Business Rule Service indirectly when a script attached to the runtime event calls Business Rule Service.
  - **Indirect invocation by a *pre* runtime event.** If the script is attached to a *pre* runtime event, such as PreWriteRecord, then:
    - If the rule module ever acts on a new record, then GetMoredata=N is mandatory. The script must construct and provide the data for the new record. GetMoreData=Y cannot be used because there is no existing record in the database from which to get the needed data.
    - If the rule module acts on existing records, then GetMoredata=N is highly recommended. GetMoredata=Y should not be used if there is any possibility that the rule should act on new, unwritten data because GetMoreData=Y forces Business Rule Service to act only on existing data in the database. Thus incorrect data could be evaluated.

- **Indirect invocation by a *post* runtime event.** If the script is attached to a *post* runtime event, such as WriteRecord, then the script can define GetMoreData=Y. There is no danger that the wrong data is provided to Business Rule Service because a *post* runtime event acts only on data that is already written to the database.

  NOTE: Setting GetMoreData=N is also allowed for this scenario.

- **Other scripted invocations.** Business Rule Service may be invoked by scripts in other contexts, such as a scripted business service or a script attached to a Siebel Workflow step.
  - If the script or workflow writes a record to the database before invoking Business Rule Service to act on the record, then the script may set GetMoreData=Y. There is no danger that the wrong data is provided to Business Rule Service because there is no choice to be made between written and new, unwritten data.

    NOTE: Setting GetMoreData=N is also allowed for this scenario.

  - If the script or workflow does not finish writing a record to the database before calling Business Rule Service, then the script must set GetMoreData = N and provide the data to Business Rule Service.

    GetMoredata=Y cannot be used if there is any possibility that the rule should act on new, unwritten data because GetMoreData=Y causes Business Rule Service to act only on existing data in the database. Thus incorrect data could be evaluated.

Setting the PerformAction Input Property

The Perform Action input property determines the entity that is responsible for performing actions, such as setting a field value, when a rule is executed.

- If PerformAction = Y, then Business Rule Service is responsible for performing an action.
- If PerformAction = N, then the caller of the Business Rule Service, such as a scripted runtime event or a script in a Siebel workflow, must take the responsibility for performing an action.

The following guidelines specify when setting PerformAction=Y is allowed and when it is mandatory or highly recommended to set PerformAction=N.

- **Direct invocation by a runtime event.** A runtime event invokes Business Rule Service directly when you declaratively configure the runtime event in the Administration - Runtime Events screen. When Business Rule Service is invoked directly by a runtime event, PerformAction = Y by default and cannot be modified.
- **Indirect invocation by a runtime event.** A runtime event invokes Business Rule Service indirectly when a script attached to the runtime event calls Business Rule Service.
  - **Indirect invocation by a *pre* runtime event.** If the script is attached to a *pre* runtime event, such as PreWriteRecord, then:
    - If the rule module ever acts on a new record, then PerformAction=N is mandatory. The script must perform any actions based on the output property set that is returned by Business Rule Service. If PerformAction = Y, then an error message is thrown.

        - If the rule module acts on existing records, then you can use PerformAction=Y, but it is highly recommended that you use PerformAction=N instead. If you use PerformAction=Y, then you must be careful that your rules do not result in an infinite loop.

          For example, assume a PreWriteRecord event calls a script which invokes Business Rules Service with PerformAction=Y. A user clicks to save a record, thereby invoking the PreWriteRecord event. The script calls Business Rule Service, which sets a field value. Because a field is modified, the PreWriteRecord event is invoked again, Business Rule Service sets the field again, and so on.

          To remedy the possible infinite loop that could be caused by a rule such as:
          `if an account's name contains "ABC Company" then set "Location" to "San Jose"`
          instead write a rule that does not unconditionally set a field value, such as:
          `if an account's name contains "ABC Company" and the account’s location is unknown then set "Location" to "San Jose"`

    - **Indirect invocation by a *post* runtime event.** If the script is attached to a *post* runtime event, such as WriteRecord, and the event acts on existing records, then you can use PerformAction=Y, but it is highly recommended that you use PerformAction=N instead. If you use PerformAction=Y, then you must be careful that your rules do not result in an infinite loop.

      For example, assume a WriteRecord event calls a script which invokes Business Rules Service with PerformAction=Y. A user clicks to save a record, thereby invoking the WriteRecord event. The script calls Business Rule Service, which sets a field value. The record is then saved, triggering the WriteRecord event again, Business Rule Service sets the field again, and so on.

      To remedy the possible infinite loop that could be caused by a rule such as:
      `if an account's name contains "ABC Company" then set "Location" to "San Jose"`
      instead write a rule that does not unconditionally set a field value, such as:
      `if an account's name contains "ABC Company" and the account’s location is unknown then set "Location" to "San Jose"`

- **Other scripted invocations.** Business Rule Service may be invoked by scripts in other contexts, such as a scripted business service or a script attached to a Siebel Workflow step.

    - If the script or workflow writes a record to the database before invoking Business Rule Service to act on the record, then the script may set PerformAction=Y or to N.

    - If the script or workflow does not finish writing a record to the database before calling Business Rule Service, then:

        - You must use PerformAction=N if there is any possibility that the rule should act on new, unwritten data.

        - If the rule always acts on existing records, then you can use PerformAction=Y or N.

About the Output Property Set of Business Rule Service

The Business Rule Service business service generates an output property set that includes a record of actions that are executed by the rules engine at runtime. These actions are of the following types:

- **Validation.** Lists *valid* or *invalid* that are set for the business component record by HaleyAuthority actions validate and invalidate.
- **Derivation.** Lists ad hoc strings that are associated with the business component record by the HaleyAuthority actions add-value, add-currency-value, and add-currency-literal.
- **SetFieldValue.** Identifies field values that are set on the business component record by HaleyAuthority actions set-field-value, set-currency-field-value, and set-currency-field-value-literal.
- **RaiseErrorText.** Lists error strings generated by the HaleyAuthority action raise-error-text.
- **SetProfileAttribute.** Lists ad hoc name-value pairs set by the HaleyAuthority action set-profile-value for use globally during one invocation of the rule module.

Programmatic access to the output property set of the Business Rules Service business service differs in different contexts.

- **Runtime event.** You cannot capture content from the output property set if you use the standard declarative means to integrate a rules module with a runtime event.

  For information on configuring a rules module to run in a runtime event, see Chapter 9, "Integrating Rules with Siebel Runtime Events.".

- **Script, Task UI step, or workflow step.** You can capture and parse the output property set with the same vehicle you used to provide the input property set, typically either standalone script or a scripted custom business service that calls the Business Rule Service business service.

  Examples of processing the output property set are provided in:

  - "Scenario for Using Rules to Provide Dynamic Navigation in a Siebel Task" on page 128
  - "Scenario for Validating Data Using Script to Invoke Rules" on page 94

To see the complete schema and examples for the Business Rule Service output property set, see "Output Property Set Schema for Business Rule Service" on page 188.

About Runtime Log Files

Execution of the Business Rule Service business service can be tracked in runtime log files that track other Siebel application execution as well.

- When using the Business Rule Service business service in a server environment, see the object manager log files.
- When using the Business Rule Service business service in a local environment (Developer client or Mobile Web Client), see the Siebel.log files that are generated in <*client root directory*>\log\.

You can vary the level of detail of Siebel log events by setting the SIEBEL_LOG_EVENTS environment variable as follows:

- 0 - Fatal
- 1 - Error
- 2 - Warning

- 3 - Information
- 4 - Detail
- 5 - Diagnostic

For detailed information on log levels and their configuration, see *Siebel System Monitoring and Diagnostics Guide*.

9 Integrating Rules with Siebel Runtime Events

This chapter provides instruction and an example for implementing rules in Siebel runtime events. The chapter contains the following topics:

Integrating Rules with a Siebel Runtime Event

You can configure a rules module to execute in response to a business component runtime event. This means of executing a rules module is useful for scenarios, such as field validation and dynamic setting of field values, in which it is not necessary to access the content in the output property set that results from calling the rule.

For a list of Siebel runtime events for which rules integration is supported, see "Siebel Runtime Events Supported by Rules" on page 184.

In general, business component runtime events can be categorized as *pre* or *post*. Typically, rule actions provide better performance when executed in *pre* runtime events instead of their corresponding *post* events. For example, implement a rule in PreWriteRecord instead of WriteRecord when possible.

To execute a rule module in any Siebel application scenario, the rule module must be called with the Business Rule Service business service.

You must complete the following tasks before you configure a runtime event to invoke rules:

1. Create the rules module in HaleyAuthority
2. Deploy the rules module from HaleyAuthority
3. Configure and activate the rules module in the Siebel client's Administration - Business Rules screen.

For information on deploying a rules module from HaleyAuthority, see "Deploying Rule Modules" on page 59.

Configuring a runtime event to invoke rules consists of:

4. Configuring an action set to invoke the Business Rule Service with appropriate rule modules
5. Associating the action set with the appropriate runtime event

For information on configuring and activating a rules module in the Administration - Business Rules screen, see "Configuring and Activating a Deployed Rules Module" on page 63.

To configure a runtime event to invoke rules

1. From the Site Map in your Siebel application, choose Administration - Runtime Events > Action Sets.

2. In the Action Sets applet, add a new action set. Use the following guidelines to enter its fields:

Field	Guideline
Name	You may want to give this a name that contains the name of the rules module that it executes, such as Account Validation Action Set.
Start Date / End Date	Null by default. If null, this action set becomes active when the Active field is set to TRUE. If a date is set in either of these fields, it restricts the dates for which the action set is active.
Active	Checked (TRUE) by default. If set to TRUE, the action set is active, subject to any restrictions in the Start Date and End Date fields. If not set to TRUE, the action set is not active.
Enable Export	Checked (TRUE) by default.

3. In the lower Actions applet, add a new action. In the bottom More Info form applet, use the following guidelines to enter its fields:

Field	Guideline
Name	Enter an arbitrary descriptive name.
Action Type	Required. Select BusService from the pick list.
Sequence	Required. ■ If there is more than one action in this action set, then enter integer sequence numbers in the order that you intend actions to execute. ■ If this is the only action for this action set, enter any simple integer, such as 1.
Active	Checked (TRUE) by default. If set to TRUE, the action is active, subject to any restrictions in the Start Date and End Date fields. If not set to TRUE, the action set is not active, even if the action set is active.

Field	Guideline
Start Date / End Date	Null by default. If null, this action becomes active when the Active field is set to TRUE. If a date is set in either of these fields, it restricts the dates for which the action is active, within the restrictions determined by the Start Date and End Date fields of the action set.
Business Service Name	Enter Business Rule Service (case sensitive). This business service executes a rules module. It is seed data in the Siebel Repository.
Business Service Method	Enter RunRules (case sensitive, no space).
Business Service Context	Enter the names of the rule modules that this action executes. To specify more than one rule module, separate their names with commas. **NOTE:** If you specify more than one rule module, then all of the specified rule modules must have the same business object. Each specified rule module must be already configured and activated using the procedure "To configure and activate a rules module" on page 63.

4 Click Events in the link bar. In the resulting Events applet, add a new event record to which you associate the action set you created. Use the following guidelines to enter its fields.

For more information about implementing runtime events, see *Siebel Personalization Administration Guide*.

Field	Guideline
Name	Not required.
Sequence	Required. ■ If this is not the only event that can trigger, then provide an integer sequence number for this and other such events to define the order you intend the events to be processed. ■ If this is the only event that can trigger, then provide a simple integer sequence number, such as 1.
Object Type	Required. Select BusComp from the pick list. A HaleyAuthority rules module cannot apply to Application or Applet objects.
Object Name	Required. From the list of values, select the business component on which this event is triggered.

Field	Guideline
Event	Required. Select the applicable event from the list of values.
Subevent	Required for some events. For example, the SetFieldValue event requires the name of the field. For information about events that require entering the Subevent field, see *Siebel Personalization Administration Guide*.
Conditional Expression	Adds additional control. If the conditional expression evaluates to TRUE, then the action is triggered when the event occurs.
Action Set Name	Required. From the list of values, select the action set that you defined in Step 2.

5 Save the event record.

For information about Business Rule Service argument requirements for direct and indirect calling of Business Rule Service by a runtime event, see “Setting the GetMoreData Input Property” on page 69 and “Setting the PerformAction Input Property” on page 70.

Scenario for Using Rules to Validate Data at Runtime

This topic provides an end-to-end example in which a runtime event is configured to invoke rules module to:

- Validate a new or modified service request record before it is written to the database.
- Auto-populate some fields.

Each of the following sections applies to one job role on the Siebel project that is responsible for completing the implementation tasks in that section.

The two job roles are:

- **Business analyst or business developer.** This individual is responsible for specifying the business process flow, the business logic requirements, and the UI requirements.
- **Siebel configuration developer.** This individual implements the requirements provided by the business analyst or business developer by performing configuration tasks in Siebel Tools, HaleyAuthority, and in the Siebel application.

The implementation process includes the following subprocesses and top-level tasks, by role.

- **Design process.** The tasks in this process are done by the business analyst or business developer:

1 Specifying Business Logic Requirements

- **Development process.** The tasks in this process are done by the Siebel configuration developer:

2 Adding Business Logic

3. Testing the Rule Modules in HaleyAuthority
4. Deploying and Activating Rule Modules
5. Configuring and Activating the Runtime Event
6. Testing Rules Modules in the Siebel Application

Specifying Business Logic Requirements

This topic is a task in the design process that is listed in "Scenario for Using Rules to Validate Data at Runtime" on page 78.

A PreWriteRecord event provides the following validation actions for a new or modified service request record:

- Disallow a service request with no description and provide an explanation to the user.
- Disallow a service request with no associated account and provide an explanation to the user.
- Disallow a service request with no associated contact and provide an explanation to the user.
- If a service request does not have an area, display a message requesting an area entry.
- If a service request has an activity plan that does not have a name, display a message to make the user aware.

  NOTE: This rule could be also be implemented in a PreWriteRecord event on the Activity Plan business component, so that when a user creates or modifies an activity plan, the user is warned to assign a name. Implementing this rule on the Activity Plan business component is appropriate if it should apply to activity plans in all contexts; that is, activity plans associated with agreements, accounts, and so on, as well as with service requests.

A WriteRecord event sets several field values for a new or modified service request record:

- If a service request has an organization whose name begins with *PCS* and the service request has no primary organization then set the organization of the service request to *Default Organization*.
- If a service request's commit time is not set, then set the commit time to the current time.
- If a service request's reproduce is set to *Always*, then set the priority of the service request to *low*.
- If a service request's reproduce is not set, then set the reproduce to *Always*.

A PreWriteRecord event sets a field value for a new or modified activity that is associated with a service request:

- If an activity is associated with a service request, and the status of the activity is *Unscheduled*, then change the status of the activity to *Alerted*.

  This rule should execute correctly in both the Service Requests screen and in the Activities screen.

  NOTE: The intent of the business rule in this case is to insure that any activity with a status of *Unscheduled* that is associated with a service request should have its status changed to *Alerted*. You could implement this rule in a PreWriteRecord event on the Service Request business component instead of on its child Action business component. However, if you implement this rule on the Service Request business component, then an activity can be assigned to a service request without the service request being subsequently saved. In such a scenario, the activity's *Unscheduled* status is not set to *Alerted*. Thus this rule is written on the Action business component so that the Status field is set appropriately whenever the condition is satisfied.

  For guidelines on writing rules on child business components, see "Writing Rules that Involve Parent and Child Business Components" on page 53.

Adding Business Logic

This topic is a task in the development process that is listed in "Scenario for Using Rules to Validate Data at Runtime" on page 78.

The business logic requirement is implemented in a PreWriteRecord runtime event and a WriteRecord runtime event, each of which makes a call to the Business Rule Service business service.

You must do the following tasks to add the business logic. Except as noted, these tasks are typical for creating and deploying rules from HaleyAuthority.

- Creating a Knowledge Base, if necessary. This task is only done once for each environment.
- Importing Siebel Objects, including establishing connectivity to the Master Repository and runtime database on the first import.
- Organizing Business Logic Into Rule Modules and Submodules.

Creating a Knowledge Base

You can use an existing knowledge base for this example or create a new knowledge base, if no knowledge base is associated with the target repository and Siebel database.

When you launch HaleyAuthority, choose an existing knowledge base or choose to create a new knowledge base. For this example, you create a new Access knowledge base that is associated with a Siebel sample database. The new Access knowledge base must be the only knowledge base associated with the target repository and database.

To create the knowledge base

1 Launch HaleyAuthority.

   - By default, launch Siebel Business Rules Developer from the Windows program group that contains Siebel Tools.

2 In the HaleyAuthority dialog box, select New knowledge base file (*.akb).

   NOTE: For this example, you create an Access knowledge base and connect to a Siebel sample database. For information on using a non-Access database, see "Setting Up Your Knowledge Base" on page 38.

3 Name the knowledge base, for example Test, and click Save.

   HaleyAuthority opens with the knowledge base name in the title bar.

Importing Siebel Objects

You import Siebel object definitions - business objects, business components, and fields - from the Siebel Master Repository in order to create rules modules that reflect the business logic. The business logic for this example is based on the Service Request business object and the Service Request business component. Several child business components of Service Request and fields in the various business components are required to be imported to implement the business logic.

On your first import of Siebel Objects into the knowledge base, you provide connectivity parameters to the Master Repository and runtime database.

To import Siebel objects and define connectivity

1 In HaleyAuthority, choose File > Import > Siebel Object.

   Siebel Object Importer Wizard launches.

2 From the Siebel Object Importer Wizard Welcome screen, click Next.

3 In the Login screen, enter database connection parameters, then click Next. For this example, a local database is used for both the repository database and as the runtime database. You may want to use the Siebel sample database, sse_samp.dbf, instead.

   - Specify (or browse to) the Siebel application cfg file (for example, uagent.cfg or siebel.cfg) which points to the sample database. It is typically located in the \BIN*LANG* directory in your Siebel developer client's root directory, where *LANG* is the language code (for example, ENU).
   - Enter the login credentials to the sample database.
   - Choose Sample as the datasource.
   - For the runtime data connection, also choose Local mode.

4 In the Pick Task screen, click the Import Siebel objects radio button, then click Next.

5 From the Pick Business Object screen, select the Service Request business object from the picklist, then click Next.

6 In the Pick Business Components screen, click the checkbox for the Service Request business component. Expand the Service Request business component hierarchy as necessary and also select the Action and Activity Plan business components, then click Next.

7 On subsequent screens of the Pick Business Component Fields dialog box, select the following fields. In the Available Fields list, select fields, then click the right arrow to move the selected fields to the Selected Fields list, then click Next.

Business Component	Fields
Service Request	Area
	Contact Last Name
	Organization
	Priority
	Account
	Commit Time
	Reproduce
	Description
Action	Status
	Activity SR Id
	NOTE: Activity SR Id is the foreign key to the parent service request in the Service Request/Action link.
Activity Plan	Name

8 The Finish screen displays the hierarchy of the business object, business component, and fields to be imported. Confirm to import this hierarchy by clicking Finish.

9 Click No when prompted whether to perform another task.

The Output screen displays a log of the concepts, relations, and dictionary objects that are automatically created in HaleyAuthority.

For more detailed information about importing Siebel objects, see "Using Siebel Object Importer" on page 47.

If you expand the concepts hierarchy, you see new concepts and phrasings that you can use with HaleyAuthority's natural English language editor to reference the imported business components and their fields with your rule statements. These phrasings are generated automatically by the Object Importer and represent the relations between the business components that are imported and between business components and fields that are imported. The phrasings are all of the form A *business component name* has a *business component name* or A *business component name* has a *field name*; for example, *a service request has a description*.

Organizing Business Logic Into Rule Modules and Submodules

You now create rule modules to represent your business logic. For this example there are two rule modules, one for the PreWriteRecord event and one for the WriteRecord event.

To create the rule modules

1. In HaleyAuthority, navigate to the Module Explorer by clicking its tab at the bottom of the screen.
2. Select Modules & Statements, then choose Object > Add a module, and name the new module Service Request Validation - PreWriteRecord.
3. Repeat Step 2 to create a module named Service Request Validation - WriteRecord and a module named Service Request Activity - PreWriteRecord.

Now you add statements to the modules to execute your business logic.

To add statements to the modules

1. Select the Service Request Validation - PreWriteRecord module and choose Object > Add a statement.
2. In the Edit Statement dialog box, type in the following statement or pick its components (except quoted strings, which you must type in) from the Words picklist, then click OK. Do not capitalize any words, except those that are in quoted strings.

   ```
   if a service request does not have an account then invalidate the service request
   with "A service request must have an account."
   ```

   NOTE: The period is part of the text to be displayed. It must be inside the double quotes. A period cannot be part of the HaleyAuthority statement syntax, so a period cannot be outside of quotes.

3. Verify that your statement is understood by HaleyAuthority. If it is not understood, a question mark is displayed at the beginning of the statement when it is displayed in the Module Explorer. If your statement is not understood, choose Object > Reparse statement. If the statement is still not understood, choose Object > Edit, and make changes so that the statement matches the direction in this procedure.
4. Repeat Step 1 through Step 3 to add the following statements to the Expense Validation module:

   ```
   if a service request does not have a contact last name then invalidate the service
   request with "A service request must have a contact."

   if a service request does not have a description then invalidate the service request
   with "A service request must have a description."

   if a service request does not have an area then raise an error as "Enter the area
   for the service request."

   if a service request has an activity plan that does not have a name then raise an
   error as "This service request has an activity plan that does not have a name."
   ```

5. Select the Service Request Validation - WriteRecord module and repeat Step 2 through Step 3 to add the following statements:

   ```
   if a service request has an organization and the organization's name begins with
   "PCS" and the service request does not have a primary organization then set
   "Organization" of the service request to "Default Organization"

   if a service request's commit time is unknown then set "Commit Time" of the service
   request to now
   ```

```
if a service request's reproduce is "Always" then set "Priority" of the service
request to "4-Low"
```

```
if a service request's reproduce is unknown then set "Reproduce" of the service
request to "Always"
```

6 Select the Service Request Activity - PreWriteRecord module and repeat Step 2 through Step 3 to add the following statement:

```
if the status of an action is "Unscheduled" then set "Status" of the action to
"Alerted"
```

Right click the statement and choose Is applicable > if, and add the applicability statement:

```
an action has an activity sr id that is not unknown
```

NOTE: Testing for a field being not unknown tests for the field being not null; that is, that the field has a value.

Testing the Rule Modules in HaleyAuthority

You can use the test harness in HaleyAuthority to test your rules in a simulation environment before deploying the rule modules.

For instruction on using the test harness, see "Using the HaleyAuthority Test Harness" on page 56.

Testing your rules in HaleyAuthority includes the following tasks:

- Create a test case.
- Create example instances of the entities that the rule module processes.
- Run the test case.
- Examine the test results.
- Modify the data and rerun the test case. (optional)

To create a test case

1 In HaleyAuthority, choose View > Tabs > Tests & Cases.

The Tests & Cases window displays.

2 Right click the Test Cases folder and choose Add a test case.

3 In the Edit Test Case dialog box, enter a name for the test case in the Description field; for example, Service Request Validation - PreWriteRecord test, then click OK.

Create the instances that serve as simulated data on which the rules act. For this example, first create a simple service request with no other associated data.

To create an example instance for the test case

1. Choose View > Tabs > Concepts.

   The Concepts window displays.

2. Locate and right click the service request entity, then choose Add > an instance or example.
3. In the instance dialog box, make the following entries, then click OK:
   - A label for the instance; for example, example_service_request_1.
   - Click the *This is an example for testing purposes only* checkbox.

   The Instances folder under service request now contains the example_service_request_1 instance.

4. Right click the example_service_request_1 instance and choose Test cases, click the Service Request Validation - PreWriteRecord test checkbox, then click OK.

Note that the example_service_request_1 instance has no service request items or field values associated with it.

To run the test case

1. Choose View > Tabs > Tests & Cases.

   The Tests & Cases window displays.

2. Locate and right click the Service Request Validation - PreWriteRecord test test case and choose Test case.
3. In the HaleyAuthority Enterprise dialog box, click Yes to deploy (or redeploy) your logic.
4. In the Deploy Configuration dialog box, make the following entries, then click OK.
   - Confirm the following checkboxes are checked: Deployment, model, measures, unauthoredRules, and unauthoredDiagnostics.
   - Check the checkbox for Service Request Validation - PreWriteRecord, and leave the checkboxes for all other modules unchecked.
   - Enter or browse to choose a directory for the deployed module; for example, D:\temp.

     NOTE: This directory has no persisting value, and does not affect the eventual runtime database to which the module is deployed.

   The test case runs.

The test case provides the following output:

- The Output window provides a log of the statements that were executed. These include unconditional action statements, conditional statements for which the if clause is true, and applicability statements that are true.

  NOTE: Siebel-specific functions (imported on your initial import with Object Importer) result in a *Loaded undefined function 'function name'* line. However, the output indicates that the simulated test executes those functions.

- An entry of the form Service Request Validation - PreWriteRecord test on <*date and time*> is added to the Test Results folder under Service Request Validation - PreWriteRecord test. Expand this folder to see the statements that were executed:

  ```
  1 if a service request does not have a contact last name then invalidate the service
  request with "A service request must have a contact."
  ```

  ```
  1 if a service request does not have an account then invalidate the service request
  with "A service request must have an account."
  ```

  ```
  1 if a service request does not have a description then invalidate the service
  request with "A service request must have a description."
  ```

  ```
  1 if a service request does not have an area then raise an error as "Enter the area
  for the service request."
  ```

  The condition for the only other statement in the module, that a service request has an activity plan without a name, is not satisfied by the example instance. The instance does not have an activity plan.

To test other conditions of your rule module, modify your example instances and rerun your test case. Alternatively you can create separate example instances, separate test cases, or both. For this example, modify example_service_request_1 to have an activity plan and some field values to test. You must set properties for your service request instances and for an activity plan instance.

To create example instances with properties

1. Create an example instance of activity plan by following the instructions in "To create an example instance for the test case" on page 85. Name the instance, for example, example_activity_plan, and associate it with the Service Request Validation - PreWriteRecord test test case.
2. Right click on example_service_request_1 and choose Properties, then click the Facts tab.
3. Click the New icon and select example_service_request_1 has an account. In the Account field, enter *ABC Company*, then click OK.
4. Similarly, add the following properties to example_service_request_1. In some cases, you will choose values from a pick list.

Property	Value
has an activity plan	example_activity_plan
has an contact last name	Lee

Rerun Service Request Validation - PreWriteRecord test simulation. The test results correctly list the following statements as being executed:

```
1 if a service request does not have a description then invalidate the service
request with "A service request must have a description."
```

```
1 if a service request does not have an area then raise an error as "Enter the area
for the service request."
```

```
1 if a service request has an activity plan that does not have a name then raise an
error as "This service request has an activity plan that does not have a name."
```

Vary your instance properties and rerun the test case to test other aspects of your rule module.

Create example instances and test cases similarly to test the Service Request Validation - WriteRecord and Service Request Activity - PreWriteRecord modules.

Deploying and Activating Rule Modules

You must deploy the Service Request Validation - PreWriteRecord and Service Request Validation - WriteRecord modules to a runtime environment.

To deploy the rule module

1. In HaleyAuthority, choose Tools > Siebel Deployment.
2. In the Siebel Deployment dialog box, select the Service Request Validation - PreWriteRecord and Service Request Validation - WriteRecord modules from the List of Modules list, check the Update model checkbox, then click OK.

   This step logs you into the runtime database and deploys the rules definitions to the runtime tables.

To activate the rule module

1. Log into the Siebel application in the same runtime environment to which you deployed the rule modules.
2. Navigate to Business Rules Administration > Rule Module List view.

   In the Rules Modules applet, the Service Request Validation - PreWriteRecord, Service Request Validation - WriteRecord, and Service Request Activity - PreWriteRecord modules are listed.
3. Select the Service Request Validation - PreWriteRecord module and set the Business Object to Service Request.
4. In the Rule Module Relation applet, create new records by picking the following values, respectively, in the Business Components field picklist.

Parent Business Component	Business Component
No entry	Service Request
Service Request	Activity Plan

5. In the Rules Modules applet, click Activate.

6 Repeat Step 4 through Step 5 for the Service Request Validation - WriteRecord module. In the Rule Module Relation applet, create new records for the following business components:

Parent Business Component	Business Component
No entry	Service Request
Service Request	Action
Service Request	Organization
Service Request	Primary Organization

7 Repeat Step 4 through Step 5 for the Service Request Activity - PreWriteRecord module. In the Rule Module Relation applet, create a new record as follows:

Parent Business Component	Business Component
No entry	Service Request
Service Request	Action

Configuring and Activating the Runtime Event

You must associate the Service Request Validation - PreWriteRecord, Service Request Validation - WriteRecord, and Service Request Activity - PreWriteRecord rule modules with runtime events.

To configure a runtime events to invoke rules

1 From the Site Map in your Siebel application, choose Administration - Runtime Events > Action Sets.

2 In the Action Sets applet, add a new action set. Use the following guidelines to enter its fields:

Field	Guideline
Name	Service Request Validation - PreWriteRecord Action
Start Date / End Date	Accept the default value null
Active	Checked (TRUE) by default.
Enable Export	Checked (TRUE) by default.

3 In the lower Actions applet, add a new action. In the lower More Info form applet, use the following guidelines to enter its fields:

Field	Guideline
Name	Service Request Validation - PreWriteRecord Rules
Action Type	Required. Select BusService from the pick list.
Sequence	Required. Enter 1.
Active	Required. Checked (TRUE) by default.
Start Date / End Date	Null by default.
Business Service Name	Enter Business Rule Service (case sensitive). This business service executes a rules module. It is seed data in your Siebel Repository.
Business Service Method	Enter RunRules (case sensitive, no space).
Business Service Context	Enter Service Request Validation - PreWriteRecord, the name of the rule module that this action executes. **NOTE:** The rule module name must match the rule module name in HaleyAuthority. The comparison is case-sensitive and space-sensitive.

4 Click Events in the link bar. In the resulting Events applet, add a new event record to which you associate the action set you created. Use the following guidelines to enter its fields.

For more information about implementing runtime events, see *Siebel Personalization Administration Guide*.

Field	Guideline
Name	(optional) Enter Service Request Validation - PreWriteRecord Event.
Sequence	Required. Enter 1.
Object Type	Required. Select BusComp from the pick list.
Object Name	Required. From the list of values, select Service Request, the top-level business component.
Event	Required. Select PreWriteRecord.
Subevent	no entry necessary
Conditional Expression	no entry necessary
Action Set Name	Required. From the list of values, select Service Request Validation - PreWriteRecord Action, the action set that you defined in Step 2.

5. Save the event record.
6. Repeat Step 1 through Step 5 to create the first WriteRecord event. Make duplicate entries, with the following exceptions:
   - Action set Name: Service Request Validation - WriteRecord Action
   - Action Name: Service Request Validation - WriteRecord Rules
   - Action Business Service Context: Service Request Validation - WriteRecord
   - Event Name: Service Request Validation - WriteRecord Event
   - Event: WriteRecord
   - Event's Action Set Name: Service Request Validation - WriteRecord Action
7. Repeat Step 1 through Step 5 to create another PreWriteRecord event. Make duplicate entries, with the following exceptions:
   - Action set Name: Service Request Activity - PreWriteRecord Action
   - Action Name: Service Request Activity - PreWriteRecord Rules
   - Action Business Service Context: Service Request Activity - PreWriteRecord
   - Event Name: Service Request Activity - PreWriteRecord Event
   - Event Object Name: Action
   - Event: PreWriteRecord
   - Event's Action Set Name: Service Request Activity - PreWriteRecord Action

Testing Rules Modules in the Siebel Application

You can test the runtime events in the Siebel application to confirm their results.

To test the runtime events

1. Log into the Siebel application in the same runtime environment to which you deployed the rule module.

   NOTE: If the application is open, first log out, then log in again.

2. Navigate to the Service Requests List view in the Service Requests screen.
3. Create new service request records or edit existing records. Create and associate activity and activity plan records with service requests to meet the criteria of the various rules written against the Service Request business component, then check to verify that:
   - Appropriate messages display.
   - If you try to save the record with disallowed data, the correct message displays and the record is not saved until all criteria for validation are met.
   - Fields are auto-populated as designed.

4. While in the Service Requests List view, click on a service request number to drill down into the service request.
5. In the resulting view, click on the Activities tab.
6. In the resulting Activities applet, create a new activity or edit an existing activity. Set its status to *Unscheduled* and save the record.

   Confirm that the status changes to *Alerted* after saving.
7. Test the runtime event similarly in the Acitivities List view of the Activities screen. Test activities with and without an entry in the SR # field.

Best Practices for Integrating Rules with Siebel Runtime Events

Table 7 provides best practices to follow when you integrate rules with runtime events.

Table 7. Best Practices for Integrating Rules with Siebel Runtime Events

Guideline	Explanation
To optimize performance: When invoking rules on runtime events, write rules for *pre* events, such as PreWriteRecord, instead of the corresponding *post* events. This guideline is especially true when the rule invokes actions, such as *set field value*.	Writing for *pre* events improves both database server performance and application server performance.
To optimize performance: Minimize the number of invocations of the Business Rule Service business service.	Every rule module invocation incurs a CPU cost.
To optimize performance: It is preferable to implement a large number of rule statements by: ■ Invoking the Business Rule Service business service a lesser number of times with rule modules that contain a greater number of rules, instead of ■ Invoking the Business Rule Service business service a greater number of times with rule modules that contain a lesser number of rules	In general, invocations of the Business Rule Service business service provide a much greater impact to performance than large rule modules.

10 Implementing Rules in Scripts

This chapter provides instruction and an example for implementing rules in script. The chapter contains the following topics:

Implementing a Rules Module to Execute From Script

You can invoke the Business Rule Service business service from scripts. A typical scenario in which to use a script to invoke Business Rule Service is when you need to process the output from Business Rule Service to make further decisions.

Using content in the output property set of Business Rule Service allows decisions to be made programmatically based on the results of rules execution. Such decision-making strategies are powerful options in applications such as Siebel Workflow and Task-Based UI flows.

For information on implementing rules in workflows and tasks, see Chapter 11, "Implementing Rules in Siebel Workflows" and Chapter 12, "Implementing Rules in Siebel Task-Based UI," respectively.

You must complete the following tasks before you create script to invoke rules:

- Create rule modules in HaleyAuthority.
- Deploy the rule modules from HaleyAuthority.
- Configure and activate the rule modules in the Siebel application's Administration - Business Rules screen.

Creating a script to invoke rules consists of the following tasks:

- Identify the business component event with which the script is associated.
- Declare a service variable that contains the Business Rule Service business service.
- Declare input and output property sets for Business Rule Service.
- Build the input property set for Business Rule Service.
  - If you define the BusCompList input property explicitly, then:
    - Identify the business components and fields in the hierarchy.
    - Build the business component-field hierarchy; that is, define the parent/child relationships.

- If you define the BusCompList input property implicitly, then provide the top-level business component only to the BusCompList input property.

  NOTE: To define BusCompList implicitly, you must set the GetMoreData input property to Y.

- Fill in the data for the other three input properties.

  For requirements for setting the GetMoreData and PerformAction input properties, see "Setting the GetMoreData Input Property" on page 69 and "Setting the PerformAction Input Property" on page 70.

For detailed information about the input property set for Business Rule Service and to see a sample input property set, see "About the Input Property Set to Business Rule Service" on page 66.

- Invoke the Business Rule Service business service
- Process Business Rule Service output data, if any output data must be accessed.

  For detailed information about the output property set of Business Rule Service, and to see a sample output property set, see "About the Output Property Set of Business Rule Service" on page 71.

Scenario for Validating Data Using Script to Invoke Rules

This topic provides an end-to-end example in which a script:

- Invokes the Business Rule Service business service with an appropriate rule module
- Processes the output from Business Rule Service

The rule module contains rules that validate a new or modified account record before it is written to the database.

Each of the following sections applies to one job role on the Siebel project that is responsible for completing the implementation tasks in that section.

The two job roles are:

- **Business analyst or business developer.** This individual is responsible for specifying the business process flow, the business logic requirements, and the UI requirements.
- **Siebel configuration developer.** This individual implements the requirements provided by the business analyst or business developer by performing configuration tasks in Siebel Tools, HaleyAuthority, and in the Siebel application.

The implementation process includes the following subprocesses and top-level tasks, by role.

- **Design process.** The tasks in this process are done by the business analyst or business developer:

1. Specifying Business Logic Requirements

- **Development process.** The tasks in this process are done by the Siebel configuration developer:

2. Adding Business Logic
3. Deploying and Activating Rule Modules
4. Creating the Script
5. Testing the Rules Module

Specifying Business Logic Requirements

This topic is a task in the design process that is listed in "Scenario for Validating Data Using Script to Invoke Rules" on page 94.

When a new account record is created or an existing account record is updated, the following validation must occur before the record is written to the database:

- Disallow an account name that contains *internal*.
- Require that the account location is entered.

Adding Business Logic

This topic is a task in the development process that is listed in "Scenario for Validating Data Using Script to Invoke Rules" on page 94.

The business logic requirement is implemented in a server script associated with a PreWriteRecord event on the Account business component that makes a call to the Business Rule Service business service.

You must do the following tasks to add the business logic. Except as noted, these tasks are typical for creating and deploying rules from HaleyAuthority.

- Creating a Knowledge Base, if necessary. This task is only done once for each environment.
- Importing Siebel Objects, including establishing connectivity to the Master Repository and runtime database on the first import.
- Organizing Business Logic Into Rule Modules.

Creating a Knowledge Base

You can use an existing knowledge base for this example or create a new knowledge base, if no knowledge base is associated with the target repository and Siebel database.

When you launch HaleyAuthority, choose an existing knowledge base or choose to create a new knowledge base. For this example, you create a new Access knowledge base that is associated with a Siebel sample database. The new Access knowledge base must be the only knowledge base associated with the target repository and database.

Follow the procedure To create the knowledge base on page 80 to create a new knowledge base.

Importing Siebel Objects

You import Siebel object definitions - business objects, business components, and fields - from the Siebel Master Repository in order to create rules modules that reflect the business logic. The business logic for this example is based on the Account business object and the Account business component.

If you created a new knowledge base, then you provide connectivity parameters to the Master Repository and runtime database on your first import of Siebel Objects into the knowledge base.

Follow the procedure To import Siebel objects and define connectivity on page 81 to import the following Siebel objects:

- Account object
- Account business component
- Name and Location fields of the Account business component

Organizing Business Logic Into Rule Modules

You now create rule modules to represent your business logic. For this example there is one rule module.

Follow the procedure To create the rule modules on page 83 to add a rule module with the name Account Validation.

Follow the procedure To add statements to the modules on page 83 to add the following statements to the Account Validation module:

```
if an account's name contains "internal" then invalidate the account with "An account
name cannot contain 'internal.' Please reenter the account name."

if an account's location is unknown then invalidate the account with "You must
provide the location for the account."
```

CAUTION: The period in each statement is part of the quoted string. The period does not terminate the rule statement. To be understood, HaleyAuthority rule statements cannot have a terminating period.

Deploying and Activating Rule Modules

This topic is a task in the development process that is listed in "Scenario for Validating Data Using Script to Invoke Rules" on page 94.

You must deploy the Account Validation module to a runtime environment.

Follow the procedure To deploy the rule module on page 87 to deploy the Account Validation rule module.

Follow the procedure To activate the rule module on page 87 with the following data to configure and activate the Account Validation rule module.

- Set the Business Object of the Account Validation module to Account.
- In the Rule Module Relation applet, create one new record and assign:

- No entry for Parent Business Component
- Business Component = Account

Creating the Script

This topic is a task in the development process that is listed in "Scenario for Validating Data Using Script to Invoke Rules" on page 94.

The business logic requirements defined in the design process specify that the validation must be performed before the record is written to the database. Based on this requirement, a script is associated with a BusComp_PreWriteRecord () event.

CAUTION: All examples of script in this document are written for use with the ST eScript engine. To implement the examples in this document, you must first confirm that Siebel Tools is set to use the ST eScript engine.

The ST eScript engine is the default scripting engine. You can use the following procedure to see which engine you are currently set to use, or to switch from the T eScript engine to the ST eScript engine.

NOTE: If you want to revert back to the T eScript engine after enabling the ST eScript engine and modifying your code to be strongly typed, you need to undo your strongly typed code changes.

To enable the ST eScript engine for Siebel Tools

1. In Siebel Tools, choose Screens > System Administration > System Preferences.
2. In the System Preferences window, under System Preferences Name, query for Enable ST Script Engine.
3. Set the System Preference Value to TRUE.
4. Recompile your Siebel Repository.

For detailed information about the ST eScript engine, see *Using Siebel Tools* and *Siebel eScript Language Reference*.

To implement the Account Validation rules module in an eScript script

1. Launch Siebel Tools against the sample database to which you deployed the Account Validation rule module.
2. In the Object Explorer, select Business Component.
3. Select and lock the Account business component.
4. Right-click the Account business component record and choose Edit Server Scripts, then choose eScript as your scripting language.
5. Select BusComp_PreWriteRecord and add the following eScript code to that function.

```
function BusComp_PreWriteRecord ()
{
   //Declare the variable that contains the Business Rule Service business service.
   var svc:Service = TheApplication().GetService(“Business Rule Service");

   //Declare the inputs property set variable for Business Rule Service. Declare
   //child and grandchild property sets that will eventually be used to create
   //the hierarchical structure of the BusCompList input property.
   var inputs:PropertySet = TheApplication().NewPropertySet();
   var child:PropertySet = TheApplication().NewPropertySet();
   var grandchild:PropertySet = TheApplication().NewPropertySet();


   //Declare the outputs property set variable for Business Rule Service
   var outputs:PropertySet = TheApplication().NewPropertySet();


   // Fill in data in the input propertyset
   // <PropertySet RuleModuleName="Account Validation"
   //              GetMoreData="N"
   //              PerformAction="N"
   //    <BusCompList>
   //       <BusComp Id="this buscomp's id" Name="Account">
   inputs.SetProperty("RuleModuleName", "Account Validation");
   inputs.SetProperty("PerformAction", "N");
   inputs.SetProperty("GetMoreData", "N");
   child.SetType("BusCompList");


   //Note that this script is associated with PreWriteRecord for the Account business
   //component, so “this” is the Account business component.
   grandchild.SetType("BusComp");
   grandchild.SetProperty("Id", this.GetFieldValue("Id"));
   grandchild.SetProperty("Name", this.Name());

   // Add fields for Name and Location
   var nameField:PropertySet = TheApplication().NewPropertySet();
   var locationField:PropertySet = TheApplication().NewPropertySet();
   nameField.SetType("Field");
   nameField.SetValue(this.GetFieldValue("Name"));
   nameField.SetProperty("Name", "Name");
   nameField.SetProperty("Type", "DTYPE_TEXT");
   //Add the Name field as a child of the Account business component.
   grandchild.AddChild(nameField);
   locationField.SetType("Field");
   locationField.SetValue(this.GetFieldValue("Location"));
   locationField.SetProperty("Name", "Location");
   locationField.SetProperty("Type", "DTYPE_TEXT");
   //Add the Location field as a child of the Account business component
   grandchild.AddChild(locationField);

   //Add the Account business component as a child of the BusCompList property.
   child.AddChild(grandchild);
   //Finally, add the BusCompList property (now a complete hierarchy) as a child of
```

```
//the inputs property set.
inputs.AddChild(child);

// invoke Business Rule Service
svc.InvokeMethod("RunRules", inputs, outputs);

// Examine the outputs propertyset which should contain the following
var wsshell = COMCreateObject("WScript.Shell");

var nChild = outputs.GetChildCount();
var i = 0;
var resultType;
var errorText;

// If no result, then the validation passed in this example.
if (nChild == 0)
{
   wsshell.Popup("Account Validation Succeeded!");
   return (ContinueOperation);
}

// Parse and process the output property set. In this example, you only look for
// RaiseErrorText and ValidationList because you only used those actions in the
// Account Validation rule module.
for (i = 0; i < nChild; i++)
{
   resultType = outputs.GetChild(i).GetType();

   // Process RaiseErrorText result
   if (resultType == "RaiseErrorText")
   {
      errorText = outputs.GetChild(i).GetValue();
      wsshell.Popup(errorText);
      return(CancelOperation);
   }

   // Process ValidationList result
   if (resultType == "ValidationList")
   {
      wsshell.Popup("Results include ValidationList back!");
      var j = 0;
      var msg:PropertySet = TheApplication().NewPropertySet();
      var valList:PropertySet = outputs.GetChild(i);
      var valCount = valList.GetChildCount();
      var valResult;

      // Look at each Validation result
      for (j = 0; j < valCount; j++)
      {
         valResult = valList.GetChild(j).GetProperty("Result");
         if (valResult != "Valid")
         {
            var val:PropertySet = valList.GetChild(j);
            var msgCount = val.GetChildCount();
```

```
                    var k = 0;
                    for (k = 0; k < msgCount; k++)
                    {
                       wsshell.Popup(val.GetChild(k).GetValue());
                    }
                    return (CancelOperation);
                 }
              }
           }
        }

        return (ContinueOperation);
     }
```

6 Save the script.

7 Click the Check Syntax button. Make edits as necessary.

8 Right-click the Account business component and select Compile Selected Objects.

9 Restart the Siebel application (the developer client).

Testing the Rules Module

This topic is a task in the development process that is listed in "Scenario for Validating Data Using Script to Invoke Rules" on page 94.

To test your script, launch the Siebel client that points to the same datasource as the Siebel Tools client you used to write the script. Make various edits to an Account record's Name and Location fields. Popup windows display with the messages in the script if you try to save the record with either a name that contains *internal* or an empty Location field.

Best Practices for Implementing Rules in Scripts

Table 8 provides best practices to follow when you implement rules in scripts.

Table 8. Best Practices for Implementing Rules in Scripts

Guideline	Explanation
Independent of how you call the Business Rule Service business service, to capture its output data, you must provide custom script that calls the Business Rule Service business service and parses its output property set.	
To optimize performance: When directly invoking the Business Rule Service business service, such as from scripts or workflow, it is recommended to set the GetMoreData input property to *N* if you can construct the necessary data required for the BusCompList input property.	For a given business component that has been imported into HaleyAuthority, various descendant business components and fields may have been imported into HaleyAuthority. For a given rules module that acts on the business component, it is likely that only a subset of the business component's descendant business components and fields are used by the rule module. Setting GetMoreData to *N* requires you to explicitly provide the business component-field hierarchy to the BusCompList input property, but it avoids having to process the entire hierarchy of a business component when the rule module executes.
To optimize performance: Minimize the number of invocations to the Business Rule Service business service.	Every rule module invocation incurs a CPU cost.
To optimize performance: It is preferable to implement a large number of rule statements by: ■ Invoking the Business Rule Service business service a lesser number of times with rule modules that contain a greater number of rules, instead of ■ Invoking the Business Rule Service business service a greater number of times with rule modules that contain a lesser number of rules	In general, invocations of the Business Rule Service business service provide a much greater impact to performance than large rule modules.

11 Implementing Rules in Siebel Workflows

This chapter provides instruction and an example for implementing rules in Siebel workflows. The chapter contains the following topics:

Implementing a Rules Module in a Siebel Workflow

Siebel Workflow allows you to define multi-step processes that can run on a pre-defined schedule, in response to a runtime event, or when manually initiated.

As in configuring a Siebel Task UI flow to execute a rules module, the strength of calling a rules module in a Siebel Workflow is that branching logic can be maintained in a declarative form and in a location external to the Siebel application. Rules also allow a Siebel workflow to execute more complex logic without requiring script.

As with leveraging a rules module in a Siebel Task UI flow, the critical requirement for leveraging a rules module in a workflow process is that you must include a Business Service step that invokes the rules module. Your business service step typically consists of a custom business service that calls the prepackaged Business Rule Service business service.

You must provide script in your custom business service to:

- Provide input arguments to the Business Rule Service business service. These input arguments may require interpreting process properties of the business service step.

    Script to provide input properties to Business Rule Service has a standard structure. For information about Business Rule Service input properties, see "About the Input Property Set to Business Rule Service" on page 66. For information and examples of using script to build the Business Rule Service input properties, see "Implementing a Rules Module to Execute From Script" on page 93 and "Scenario for Using Rules to Provide Dynamic Navigation in a Siebel Task" on page 128.

- If you must capture output from Business Rule Service to use in a subsequent workflow step, then your script must parse the Business Rule Service output property set and assign that output into output arguments of the business service step.

  For information about Business Rule Service output properties, see "About the Output Property Set of Business Rule Service" on page 71. For information and examples of using script to capture Business Rule Service output, see "Implementing a Rules Module to Execute From Script" on page 93 and "Scenario for Using Rules to Provide Dynamic Navigation in a Siebel Task" on page 128.

The process of creating and deploying an entire workflow process requires that you complete several tasks such as planning the process flow, defining the business logic, creating the workflow, creating the business logic and integrating it with the workflow process, deploying and activating the workflow process, testing, and so on.

For a complete example of planning and implementing a workflow that uses a rules module in a batch process, see "Scenario for Using Rules in a Siebel Workflow to Do Batch Processing" on page 104.

For information on the Business Rule Service business service, see "About the Business Rule Service Business Service" on page 66. For requirements for setting the Business Rule Service input properties GetMoreData and PerformAction, see "Setting the GetMoreData Input Property" on page 69 and "Setting the PerformAction Input Property" on page 70.

For detailed information on creating and deploying Siebel workflows, see *Siebel Business Process Framework: Workflow Guide*.

Scenario for Using Rules in a Siebel Workflow to Do Batch Processing

This topic provides an end-to-end example in which a Siebel workflow is configured to do batch processing of Opportunity records. The process generates a comment in the Sales Objective field when the lead quality and probability of the opportunity meet an inconsistency threshold.

The process is run with the mobile Web client running against the sample database.

Each of the following sections applies to one job role on the Siebel project that is responsible for completing the implementation tasks in that section.

The two job roles are:

- **Business analyst or business developer.** This individual is responsible for specifying the business process flow and the business logic requirements.
- **Siebel configuration developer.** This individual implements the requirements provided by the business analyst or business developer by performing configuration tasks in Siebel Tools, HaleyAuthority, and in the Siebel application.

The implementation process includes the following subprocesses and top-level tasks, by role.

- **Design process.** The tasks in this process are done by the business analyst or business developer:

1. "Specifying Business Logic Requirements" on page 105
2. "Defining the Business Process Flow" on page 105
3. "Specifying Invocation Requirements" on page 106

- **Development process.** The tasks in this process are done by the Siebel configuration developer:

4. "Creating the Workflow" on page 106
5. "Adding Business Logic" on page 113

6 “Testing the Rule Module in HaleyAuthority” on page 116

7 “Deploying and Activating Rule Modules” on page 118

8 “Providing Calls to the Rules Engine” on page 119

9 “Testing the Workflow in Simulation Mode” on page 123

10 “Deploying, Activating, and Administering the Workflow” on page 124

NOTE: This example does not leverage output from rules to do branching in the workflow. To do such branching in a workflow, do similar configuration as is shown in “Scenario for Using Rules to Provide Dynamic Navigation in a Siebel Task” on page 128.

Specifying Business Logic Requirements

This topic is a task in the design process that is listed in “Scenario for Using Rules in a Siebel Workflow to Do Batch Processing” on page 104.

This business process looks at opportunities with Status = Accepted.

For each accepted opportunity, sales management expects that:

- If the lead quality is excellent, then the probability of completing the sale is at least 70%.
- If the lead quality is very high, then the probability of completing the sale is at least 50%.

If either of these conditions is violated, then either the lead quality or the probability of the opportunity must be changed. A comment is generated in the Sales Objective text box that states that these fields should be reconciled.

The business logic refers to field captions used in Opportunity-based applets - Status, Lead Quality, Probability %, and Sales Objective. However, to implement this process, you must apply the logic to fields on the Opportunity business component. You can use Siebel Tools to determine the business component fields underlying the applet fields. They are:

Applet Field Caption	Underlying Field on Opportunity Business Component
Status	Status
Lead Quality	Quality
Probability %	Primary Revenue Win Probability
Sales Objective	Description

Defining the Business Process Flow

This topic is a task in the design process that is listed in “Scenario for Using Rules in a Siebel Workflow to Do Batch Processing” on page 104.

This design task can be done in any application that supports flowcharting or other representation of process flow.

For this example, the business process is:

1. Query opportunities for Status = Accepted.
2. For each record in the result set:
   - If Quality is 1-Excellent, and Primary Revenue Win Probability is at less than 70%, then set Description to *Reconcile lead quality and probability %.*
   - If Quality is 2-Very High, and Primary Revenue Win Probability is at less than 50%, then set Description to *Reconcile lead quality and probability %.*

This process is based on the Opportunity business object and the Opportunity business component.

Specifying Invocation Requirements

This topic is a task in the design process that is listed in "Scenario for Using Rules in a Siebel Workflow to Do Batch Processing" on page 104.

This batch process can be run manually or on a preset schedule determined by a sales manager.

Creating the Workflow

This topic is a task in the development process that is listed in "Scenario for Using Rules in a Siebel Workflow to Do Batch Processing" on page 104.

To allow you to assign display names to workflow objects, confirm that the following parameter is set in the [Siebel] section of \TOOLS_ROOT\bin\LANGUAGE\tools.cfg:

```
EnableToolsConstrain = FALSE
```

If either parameter is not present, add it.

This process consists of the following tasks:

- Determine a project in Siebel Tools to associate all the metadata for this business process.
- Create a business component that is a clone of the Opportunity business component, to meet the requirements of a looping workflow process.
- Create the workflow process structure.
- Configure the workflow process steps.

Determining a Project

You can create a new project with which to associate the metadata artifacts for this business process, or you can use a pre-existing project.

If you use a pre-existing project, lock the project.

To create a new project

1. In Siebel Tools, navigate to Project in the Object Explorer.
2. Choose Edit > New Record.
3. Enter a name for the project, for example, Business Rule.
4. Click the Locked checkbox to lock the project.
5. Step off the new Project record to save the record.

Creating a New Business Component

Siebel workflow processes have the following requirements:

- The workflow process must process a record from the primary business component of the business object on which the workflow process is based.
- A looping workflow process that processes a batch of records can only loop through records that are of a non-primary business component of the business object on which the workflow process is based.

The Opportunity business component is the primary business component of the Opportunity business object, so to satisfy these requirements you create a new business component, Opportunity No Link, that is a clone of the Opportunity business component.

The new business component contains the minimum number of fields required for this business process, and those fields point to the same table columns as their counterparts in the Opportunity business component. Thus, when a field is modified in the clone, the same field is modified in the Opportunity business component record.

To create the Opportunity No Link business component

1. In Siebel Tools, verify that the Business Rule project is still locked.
2. Navigate to Business Component in the Object Explorer.
3. Choose Edit > New Record.
4. In the Business Components list, enter the following property values for the new business component. Accept default values for other properties.

Property	Value
Name	Opportunity No Link
Project	Business Rule
Class	CSSBCBase
Table	S_OPTY

5. With Opportunity No Link selected in the Business Components list, expand the Business Component object in the Object Explorer, then click Field.

6 Choose Edit > New Record to add each of the following fields to the Opportunity No Link business component. Use the following guidelines to enter each field's properties. Accept default values for other properties.

NOTE: These fields correspond to the fields with the same names on the Opportunity business component.

Name	Column	Text Length	Type
Status	STATUS_CD	30	DTYPE_TEXT
Quality	LEAD_QUALITY_CD	30	DTYPE_TEXT
Primary Revenue Win Probability	SUM_WIN_PROB	not applicable	DTYPE_NUMBER
Description	DESC_TEXT	255	DTYPE_TEXT

7 Choose Tools > Compile Selected Objects, and compile the Opportunity No Link business component to the repository that is used by your sample database (typically <*Siebel client root directory*>\OBJECTS\<*language*>\siebel.srf.

Creating the Workflow Process Structure

This workflow process:

- Accepts a record of the Opportunity business component, which is not processed.
- Queries the Opportunity No Link business component for records whose Status is *Accepted*.
- Loops through records in the query result set, applying the business logic.

To create the workflow process structure

1 In Siebel Tools, verify that the Business Rule project is still locked.

2 In the Siebel Tools Object Explorer select the Workflow Process object.

3 Choose Edit > New Record.

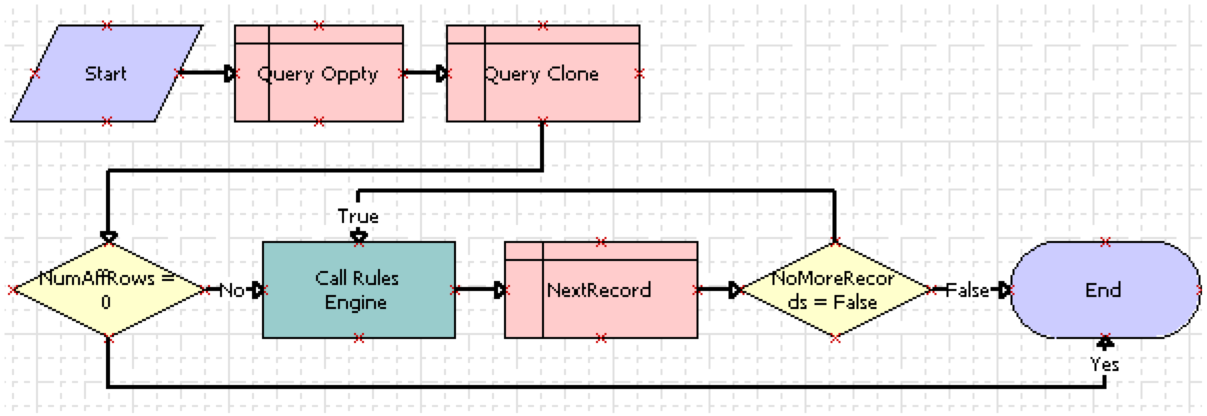
4 In the Workflow Processes list, enter the following property values for the new workflow process. Other properties' values are read only or should be left empty.

Field	Value
Process Name	Opportunity Rules Batch Processing
Workflow Mode	Service Flow
Business Object	Opportunity
Project	Business Rule

5 In the Workflow Processes list, right-click the Opportunity Rules Batch Processing workflow process and choose Edit Workflow Process. In the gridded canvas, model a workflow process similar to the following by clicking and dragging steps and connectors from the palette.

If the palette is not visible, choose View > Windows > Palette.

Note that you are simply drawing a picture of steps and connectors at this point. You configure the metadata in subsequent steps.



Enter names for steps and labels for connectors as shown.

- Click on a step and enter the name of a step in the Name field of the Properties window for the step.
- Click on a connector between steps and enter the label for the connector in the Name field of the Properties window for the connector.

6 Click anywhere on the canvas away from steps and connectors to display the Multi Value Property Window for the entire workflow process. Right click in the Process Properties window and choose New Record, then add the following process properties to the default list. Accept default values for fields not listed.

Name	Business Object	Data Type
NoRecord	Opportunity	String
NumberOfRows	Opportunity	Number
vRowId	Opportunity	String

7 Choose File > Save.

To configure the workflow process steps

1 On the canvas, select each step and connector one by one and add the following metadata. For properties not noted, accept defaults.

Step or Connector	Metadata
Start	No entries
Connector: Start to Query Oppty	No entries
Query Oppty	In the Properties window, pick from each picklist: ■ Business Component: Opportunity ■ Operation: QueryBiDirectional **NOTE:** This step does an empty query that returns all Opportunity records to provide the one required record that is described in "Creating a New Business Component" on page 107. In practice, devise a query that returns one or a few records for better performance.
Connector: Query Oppty to Query Clone	No entries

Step or Connector	Metadata
Query Clone	In the Properties window, pick from each picklist: ■ Business Component: Opportunity No Link ■ Operation: Query In Multi Value Property Window > Search Spec Input Arguments, right click and choose New Record. Pick or enter the following field values only: ■ Filter Business Component: Opportunity ■ Search Specification: [Status] = 'Accepted' **NOTE:** Single quotes around the string value and spaces around the equal sign are required. ■ Type: Literal In the Multi Value Property Window > Output Arguments, right click and choose New Record. Pick or enter the following field values only: ■ Property Name: NumberOfRows ■ Type: Output Argument ■ Output Argument: NumAffRows In the Multi Value Property Window > Output Arguments, create a second record with the following field values: ■ Property Name: vRowID ■ Type: Business Component ■ Business Component Name: Opportunity No Link ■ Business Component Field: Id
Connector: Query Clone to NumAffRows = 0	No entries
NumAffRows = 0	No entries
Connector: NumAffRows = 0 *Yes* branch	In the Properties window, verify that Type is Default.

Step or Connector	Metadata
Connector: NumAffRows = 0 *No* branch	In the Properties window, set Type to Condition. Select the connector and right-click, then choose Edit Condition. Define the following condition, and click OK: ■ Compare To = Process Property ■ Operation = Greater Than ■ Object = NumberOfRows ■ Value = 0
Call Rules Engine	You make entries for this step later in "Providing Calls to the Rules Engine" on page 119.
Connector: Call Rules Engine to NextRecord	No entries
NextRecord	In the Properties window, set Operation to NextRecord. In the Multi Value Property Window > Output Arguments, right click and choose New Record. Pick or enter the following field values only: ■ Property Name: NoRecord ■ Type: Output Argument ■ Output Argument: NoMoreRecords
Connector (NextRecord to NoMoreRecords = False)	No entries
NoMoreRecords = False	No entries
Connector: NoMoreRecords = False *True* branch	In the Properties window, set Type to Condition. Select the connector and right-click, then choose Edit Condition. Define the following condition, and click OK: ■ Compare To: Process Property ■ Operation: All Must Match (Ignore Case) ■ Object: NoRecord ■ Value: false
Connector: NoMoreRecords = False *False* branch	In the Properties window, verify that Type is Default.
End	No entries.

2 Choose File > Save.

3 In the Workflow Processes list applet, select Opportunity Rules Batch Processing, then choose Tools > Unlock Project.

Adding Business Logic

This topic is a task in the development process that is listed in "Scenario for Using Rules in a Siebel Workflow to Do Batch Processing" on page 104.

The business logic requirement is implemented in a business service step that makes a call to the Business Rule Service business service. Business Rule Service calls a rules module that applies the logic that is described in "Defining the Business Process Flow" on page 105.

You must do the following tasks to add the business logic. Except as noted, these tasks are typical for creating and deploying rules from HaleyAuthority.

- "Creating a Knowledge Base" on page 113. This task is only done once for each environment.
- "Importing Siebel Objects" on page 114, including establishing connectivity to the Master Repository and runtime database on the first import.
- "Organizing Business Logic Into Rule Modules" on page 115.
- "Deploying and Activating Rule Modules" on page 118.

Creating a Knowledge Base

For this example, if you have already deployed any rules to the runtime database you intend to use, then use your existing knowledge base that points to that runtime database. If you have not deployed any rules, you can use an existing knowledge base or you can create a new knowledge base.

To create the knowledge base

1 Launch HaleyAuthority by choosing Siebel Business Rules Developer from the Windows program group that contains Siebel Tools.

2 In the HaleyAuthority dialog box, select New knowledge base file (*.akb).

   NOTE: For this example, you create an Access knowledge base and connect to a Siebel sample database. For information on using a non-Access database, see "Setting Up Your Knowledge Base" on page 38.

3 Name the knowledge base, for example KB_1, and click Save.

   HaleyAuthority opens with the knowledge base name in the title bar.

Importing Siebel Objects

You import Siebel object definitions - business objects, business components, and fields - from the Siebel Master Repository in order to create rules modules that reflect the business logic. The business logic for this example is based on the Opportunity business object and Opportunity business component. You also based the Opportunity Rules Batch Processing workflow process on the Opportunity business object.

If this is your first import of Siebel Objects into the knowledge base, you must provide connectivity parameters to the Master Repository and runtime database.

To import Siebel objects and define connectivity

1. In HaleyAuthority, choose File > Import > Siebel Object.

   Siebel Object Importer Wizard launches.

2. From the Siebel Object Importer Wizard Welcome screen, click Next.

3. If this is your first import of Siebel objects into this knowledge base, enter repository and runtime database connection parameters in the Login screen, then click Next. For this example, the Siebel sample database is used for both the repository database and as the runtime database.

   - Specify (or browse to) the Siebel application cfg file (for example, uagent.cfg or siebel.cfg) which points to the applicable database. It is typically located in the \BIN*LANG* directory in your Siebel Mobile Web client root directory, where *LANG* is the language code (for example, ENU).

   - Enter the login credentials to the applicable database.

   - Choose Sample for the datasource.

   - For the runtime data connection, choose Local mode.

4. In the Pick Task screen, click the Import Siebel objects radio button, then click Next.

5. From the Pick Business Object screen, select the Opportunity business object from the picklist, then click Next.

6. in the Pick Business Components screen, click the checkboxes for the Opportunity and Opportunity No Link business components, then click Next.

7. The Pick Business Component Fields dialog box lists available fields for the Opportunity business component. In the Available Fields list, click to select the Description, Primary Revenue Win Probability, and Quality fields. Click the right arrow to move the selected fields to the Selected Fields list, then click Next.

   NOTE: You do not have to select the Status field. The Status field is used in a query in a workflow step, but it is not used in the rule module.

8. Similarly, for the Opportunity No Link business component, select the Description, Primary Revenue Win Probability, and Quality fields.

9. The Finish screen displays the hierarchy of the business object, business component, and fields to be imported. Confirm to import this hierarchy by clicking Finish.

10 Click No when prompted whether to perform another task.

The Output screen displays a log of the concepts, relations, and dictionary objects that are automatically created in HaleyAuthority.

For more detailed information about importing Siebel objects, see "Using Siebel Object Importer" on page 47.

After running the object importer, opportunity, opportunity no link, description, quality, and primary revenue win probability are available as concepts.

If you expand the Concepts hierarchy, you see phrasings that you can use with HaleyAuthority's natural English language editor to reference the imported business components and their fields with your rule statements. These phrasings are generated automatically by the Object Importer and represent the relations between the business components that are imported and their fields that are imported. The phrasings are all of the form a *business component name* has a *field name*; for example, *an opportunity has a description*.

Organizing Business Logic Into Rule Modules

A module is a grouping of rule statements. The business logic in this example requires that you create one module.

To create the rule module

1 In HaleyAuthority, navigate to the Module Explorer by clicking its tab at the bottom of the screen.

2 Select Modules & Statements, then choose Object > Add a module, and name the new module Opportunity Batch Processing Rules.

NOTE: Alternatively, if you used a different name for the rule module when you defined the RuleModuleName input argument for the Call Rules Engine workflow step in "To create the workflow process structure" on page 108, use that name.

3 Select the Opportunity Batch Processing Rules module, and choose Object > Add a statement.

4 In the Edit Statement dialog box, type in the following statement or pick its components, then click OK. Do not capitalize any words, except those that are in quoted strings.

```
set "Description" of an opportunity to "Reconcile lead quality and probability %."
```

NOTE: For a *set* statement, the field name must be quoted and not preceded by an article, such as *the* or *a*.

5 Select the statement you created in Step 4 and choose Object > Is applicable > if.

6 In the Edit Statement dialog box, type in the following statement or pick its components, then click OK. Do not capitalize any words, except those that are in quoted strings.

```
the quality of the opportunity is "1-Excellent" and the primary revenue win
probability of the opportunity is less than 70
```

NOTE: *1-Excellent* must be identical to the field value to which it is being compared. The comparison is case-sensitive and space-sensitive.

7 Repeat Step 5 and Step 6 to add a second applicability statement:

```
the quality of the opportunity is "2-Very High" and the primary revenue win
probability of the opportunity is less than 50
```

NOTE: If either *if* applicability condition is true, the statement executes.

8 Repeat Step 3 through Step 7 to add the following statement and applicability conditions to the module. The second set is identical to the first set, but applies to opportunity no link.

```
set "Description" of an opportunity no link to “Reconcile lead quality and
probability %.”

Applicability:

    if: the quality of the opportunity no link is "1-Excellent" and the primary
    revenue win probability of the opportunity no link is less than 70

    if: the quality of the opportunity no link is "2-Very High" and the primary revenue
    win probability of the opportunity no link is less than 50
```

Testing the Rule Module in HaleyAuthority

This topic is a task in the development process that is listed in “Scenario for Using Rules in a Siebel Workflow to Do Batch Processing” on page 104.

You can use the test harness in HaleyAuthority to test your rules in a simulation environment before deploying the rule module.

For instruction on using the test harness, see “Using the HaleyAuthority Test Harness” on page 56.

Testing your rules in HaleyAuthority includes the following tasks:

- Create a test case.
- Create example instances of the entities that the rule module processes.
- Run the test case.
- Examine the test results.
- Modify the data and rerun the test case. (optional)

To create a test case

1 In HaleyAuthority, choose View > Tabs > Tests & Cases.

The Tests & Cases window displays.

2 Right click the Test Cases folder and choose Add a test case.

3 In the Edit Test Case dialog box, enter a name for the test case in the Description field; for example, Opportunity Rules, then click OK.

Create the instances that serve as simulated data on which the rules act.

To create example instances for the test case

1. Choose View > Tabs > Concepts.

   The Concepts window displays.

2. Locate and right click the opportunity entity, then choose Add > an instance or example.
3. In the instance dialog box, make the following entries, then click OK:
   - A label for the instance; for example, example_opportunity.
   - Click the *This is an example for testing purposes only* checkbox.

   The Instances folder under opportunity now contains the example_opportunity instance.

4. Right click the example_opportunity instance and choose Test cases, click the Opportunity Rules checkbox, then click OK.

   NOTE: Alternatively, you can add all example instances to a test case by right clicking the test case in the Tests & Cases window and choosing Examples.

5. Right click on example_opportunity and choose Properties, then click the Facts tab.
6. Click the New icon and select *example_opportunity has a quality*. For the quality value, enter *1-Excellent*, then click OK.
7. Repeat Step 6 to add the following fact to the example_opportunity instance:

   example_opportunity has a primary revenue win probability

   Enter 69 as the value of the primary revenue win probability.

8. Repeat Step 2 through Step 7 to create an instance of opportunity no link, named example_opptynolink, with the following fact, but do not yet add example_opptynolink to the Opportunity Rules test case.

   example_opptynolink has a quality, with 2-Very High as the value for quality

   example_opptynolink has a primary revenue win probability, with 48 as the value of the primary revenue win probability.

This example is intended to act on one opportunity or opportunity no link record at a time, so associate only one instance at a time with the Opportunity Rules test case.

To run the test case

1. Choose View > Tabs > Tests & Cases.

   The Tests & Cases window displays.

2. Locate and right click the Opportunity Rules test case and choose Test case.
3. In the HaleyAuthority Enterprise dialog box, click Yes to deploy (or redeploy) your logic.
4. In the Deploy Configuration dialog box, make the following entries, then click OK.
   - Accept the default of the Deploy checkboxes being checked for all modules.

- Enter or browse to choose a directory for the deployed module; for example, D:\temp.

  NOTE: This directory has no persisting value, and does not affect the eventual runtime database to which the module is deployed.

  The test case runs and processes the example_opportunity instance.

The test case provides the following output:

- The Output window provides a log of the statements that were executed.
- An entry of the form Opportunity Rules on <*date and time*> is added to the Test Results folder under Opportunity Rules. Expand this folder to see the statements that were executed:

  ```
  1 if: the quality of the opportunity is "1-Excellent" and the primary revenue win
  probability of the opportunity is less than 70
  ```

  ```
  1 set "Description" of an opportunity to “Reconcile lead quality and probability %.”
  ```

You can disassociate example_opportunity and associate example_opptynolink with the Opportunity Rules test case to test another aspect of the rule module. You can create other instances to test other aspects.

Deploying and Activating Rule Modules

This topic is a task in the development process that is listed in “Scenario for Using Rules in a Siebel Workflow to Do Batch Processing” on page 104.

You must deploy the Opportunity Batch Processing Rules module to a runtime environment for testing.

To deploy the rule module

1. In HaleyAuthority, choose Tools > Siebel Deployment.
2. In the Siebel Deployment dialog box, select Opportunity Batch Processing Rules from the List of Modules list, check the Update model checkbox, then click OK.

   This step logs you into the runtime database and deploys the Opportunity Batch Processing Rules rules definitions to the runtime tables.

To activate the rule module

1. Log into the Siebel application in the same runtime environment to which you deployed the rule module.
2. Navigate to Business Rules Administration > Rule Modules List view.

   In the Rule Modules applet, Opportunity Batch Processing Rules is listed as a module.
3. Set the Business Object of the Opportunity Batch Processing Rules module to Opportunity.

4 In the Rule Module Relations applet, create two new records. For one, set the Business Component field to Opportunity, and for the other set the Business Component field to Opportunity No Link. Make no entry in the Parent Business Component field for either record.

  NOTE: There is no need to add other records because no child business components of Opportunity are included in the Opportunity Batch Processing Rules module.

5 In the Rule Modules applet, click Activate.

  The Status field of the Opportunity Batch Processing Rules module displays Active.

Providing Calls to the Rules Engine

This topic is a task in the development process that is listed in "Scenario for Using Rules in a Siebel Workflow to Do Batch Processing" on page 104.

Now that the rules module has passed a simulation test, the last task is to create and use a business service to call the rule module. You must do two configuration tasks:

- Create a business service that calls the rule module.
- Include the business service in the workflow definition.

To create a business service that calls the rule module

1 In Siebel Tools Object Explorer, select Business Service.

2 Create a new business service with the following properties:

  Name = Rules Business Call - WF

  Project = Business Rule

  Cache = TRUE

  Display Name - String Override = Rules Business Call - WF

  External Use = TRUE

3 Expand the Business Service and add a business service method with the following properties to the Rules Business Call - WF business service you created in Step 2:

  Name = Rules

  Display Name - String Override = Rules

4 Expand the Business Service Method object and add the following business service argument to the Rules business service method you created in Step 3.

Name	Data Type	Type	Storage Type	Display Name - String Override
vRowId	String	Input	Property	vRowId

5 Right-Click on the Rules Business Call - WF business service and choose Edit Server Scripts.

6 Select Service_PreInvokeMethod and enter the following script.

CAUTION: All examples of script in this document are written for use with the ST eScript engine. To implement this example, first confirm that you are set to use the ST eScript engine in Siebel Tools. For information on setting Siebel Tools to use the ST eScript engine, see "To enable the ST eScript engine for Siebel Tools" on page 97.

```
function Service_PreInvokeMethod (MethodName, Inputs:PropertySet,
Outputs:PropertySet)

{

   if (MethodName == "Rules")

   {

      var out:chars = Rules(Inputs, Outputs);

      return (CancelOperation);

   }

   return (ContinueOperation);

}
```

7 Add the following custom method to the Rules Business Call - WF business service:

```
function Rules(Inputs:PropertySet, Outputs:PropertySet)

{

   try

   {

      //Declare your Service and property sets to build your hierarchical structure.

      var svc:Service = TheApplication().GetService("Business Rule Service");

      var inputs:PropertySet = TheApplication().NewPropertySet();

      var outputs:PropertySet = TheApplication().NewPropertySet();

      var child:PropertySet = TheApplication().NewPropertySet();

      var grandchild:PropertySet = TheApplication().NewPropertySet();


      //Declare the variable that gets the row id of the current opportunity

      var RowIdInput:chars = Inputs.GetProperty("vRowId");

      //Declare variable to store results

      var valResult:chars;

      var TheResult:chars;
      //Use methods to build input property set structure.
      //Setting RuleModuleName to the module to call.
      inputs.SetProperty("RuleModuleName", "Opportunity Batch Processing Rules");

      //Setting PerformAction to "Y" because the rule statements include
      //setfieldvalue calls, which are actions.
      inputs.SetProperty("PerformAction", "Y");

      //Setting GetMoreData to "Y" because you want the rules engine to get all
      //the fields automatically, rather than passing them all explicitly.
      inputs.SetProperty("GetMoreData", "Y");

      //Populate variables to create the BusCompList > business component > field
      //hierarchy that will eventually be provided as the BusCompList input
      //property set to Business Rules Service.
      child.SetType("BusCompList");
      grandchild.SetType("BusComp");
      grandchild.SetProperty("Id", RowIdInput);
      grandchild.SetProperty("Name", "Opportunity");
      child.AddChild(grandchild);
      inputs.AddChild(child);

      //Invoke Business Rule Service.
      svc.InvokeMethod("RunRules", inputs, outputs);

   } //end try
   catch(e)
   {
      TheApplication().RaiseErrorText("Error in the Rules function " +
e.toString());

   }

   finally

   {

   }

}
```

8 Navigate back to the Business Services list, select the Rules Business Call - WF business service, and choose Tools > Compile Selected Objects.

Finish the Opportunity Rules Batch Processing workflow process definition to include the new Rules Business Call - WF business service.

To include the business service in the workflow process definition

1 In Siebel Tools, navigate to the Workflow Process object and select the Opportunity Rules Batch Processing workflow process.

2 Right-click and choose Edit Workflow Process to access the Process Designer. The status property of the Opportunity Rules Batch Processing workflow process is In Progress.

3 Select the main grid with no steps selected. With the Process Properties tab selected, right click in the Multi Value Property Window, and choose New Record to add the following process property:

Name	In/Out	Business Object	Data Type	Access Mode
vRowId	In/Out	Opportunity	String	RW

4 Select the Call Rules Engine business service step and assign the following values in the Properties window:

Property	Value
Business Service Method	Rules
Business Service Name	Rules Business Call - WF

5 With the Call Rules Engine business service step selected, click the Input Arguments tab in the Multi Value Property Window. Right-click, then choose New Record to add the following input argument:

Input Argument	Type	Property Name
vRowId	Process Property	vRowId

Testing the Workflow in Simulation Mode

This topic is a task in the development process that is listed in "Scenario for Using Rules in a Siebel Workflow to Do Batch Processing" on page 104.

You can test the workflow by simulating it in a debug instance of your runtime client.

Before testing, you may want to do the following tasks in your runtime client:

- Make the Status, Quality, Description, and Primary Revenue Win Probability fields visible in the Opportunity list applet that you are using.

  NOTE: Remember that the captions of these fields in the UI are Status, Lead Quality, Sales Objective, and Probability %.

- Edit or create some new opportunities with Status, Quality, and Primary Revenue Win Probability values that satisfy the rule conditions.

To make columns visible in a list applet

1. Right click anywhere in the list applet and choose Columns Displayed.
2. The fields are listed by their captioned names - Status, Lead Quality, Sales Objective, and Probability %. Move them from the Available Columns list to the Selected Columns list and click Save.

Before running the simulator, confirm that your Debug options are accurate and point the runtime client that you have specified elsewhere in this example. For information on setting Debug options, see *Siebel Business Process Framework: Workflow Guide*.

NOTE: The paths you must enter as Debug options must be specific to your local installation.

NOTE: The workflow simulator modifies the target database.

To simulate the workflow process

1. Confirm that the workflow process is saved.
2. If the Simulate toolbar is not visible, choose View > Toolbars > Simulate.
3. Right click Opportunity Rules Batch Processing in the Workflow Processes list and choose Simulate Workflow Process on the canvas away from steps and connectors, then choose Simulate.

   The workflow process diagram displays with the Start step highlighted.
4. Click the Start button (leftmost right arrow) on the Simulate toolbar.

   A debug runtime instance of your runtime client starts.
5. If the Watch window is not visible, choose View > Debug Windows > Watch.

   The Watch window displays data such as output argument values as you step through a workflow process. If the Watch window is not already visible, you cannot open it until you execute the Start step.

6 Click the Simulate Next button (second from left) to step through the process or click the Complete Simulation button (second from right) to execute the rest of the process. Click the Stop Simulation button (rightmost) to stop the simulation.

Check the results in the runtime client.

For additional detailed information on running the workflow simulator in Siebel Tools, see *Siebel Business Process Framework: Workflow Guide*.

Deploying, Activating, and Administering the Workflow

This topic is a task in the development process that is listed in "Scenario for Using Rules in a Siebel Workflow to Do Batch Processing" on page 104.

Before you can use the Opportunity Rules Batch Processing workflow process, you must:

- Publish the workflow process from Siebel Tools.
- Activate the workflow process in the Administration - Business Process screen of your Siebel application. Once the workflow process is activated, you can invoke it manually or set it up to run on a preset schedule.

You can also start, stop, and monitor workflow processes in your Siebel application's Administration - Business Process screen.

For information on deploying and activating workflow processes, implementing workflow policies, and monitoring workflows, see *Siebel Business Process Framework: Workflow Guide*.

Best Practices for Implementing Rules in Siebel Workflows

Table 9 provides best practices to follow when you implement rules in Siebel Workflows.

Table 9. Best Practices for Implementing Rules in Siebel Workflows

Guideline	Explanation
Independent of how you call the Business Rule Service business service, to capture its output data, you must provide custom script that calls the Business Rule Service business service and parses its output property set.	
To optimize performance: When directly invoking the Business Rule Service business service, such as from scripts or workflow, it is recommended to set the GetMoreData input property to *N* if you can construct the necessary data required for the BusCompList input property.	For a given business component that has been imported into HaleyAuthority, various descendant business components and fields may have been imported into HaleyAuthority. For a given rules module that acts on the business component, it is likely that only a subset of the business component's descendant business components and fields are used by the rule module. Setting GetMoreData to N requires you to explicitly provide the business component-field hierarchy to the BusCompList input property, but it avoids having to process the entire hierarchy of a business component when the rule module executes.
To optimize performance: Minimize the number of invocations to the Business Rule Service business service.	Every rule module invocation incurs a CPU cost.
To optimize performance: It is preferable to implement a large number of rule statements by: ■ Invoking the Business Rule Service business service a lesser number of times with rule modules that contain a greater number of rules, instead of ■ Invoking the Business Rule Service business service a greater number of times with rule modules that contain a lesser number of rules	In general, invocations of the Business Rule Service business service provide a much greater impact to performance than large rule modules.

12 Implementing Rules in Siebel Task-Based UI

This chapter provides instruction and an example for implementing rules in Siebel Task-Based UI flows. The chapter contains the following topics:

- "Implementing a Rules Module in a Siebel Task UI Flow" on page 127
- "Scenario for Using Rules to Provide Dynamic Navigation in a Siebel Task" on page 128
- "Best Practices for Implementing Rules in Siebel Tasks" on page 165

Implementing a Rules Module in a Siebel Task UI Flow

Siebel Task UI allows you to define UI tasks that help the user to complete business processes.

As in configuring a Siebel workflow to execute a rules module, the strength of calling a rules module in a Siebel Task UI flow is that branching logic can be maintained in a declarative form and in a location external to the Siebel application.

The critical requirement for leveraging a rules module in a task flow is that you must include a Business Service step that invokes the rules module. Your business service step typically consists of a custom business service that calls the prepackaged Business Rule Service business service.

You must provide script in your custom business service to:

- Provide input arguments to the Business Rule Service business service. These input arguments may require interpreting task properties of the business service step.
- If output from the Business Rule Service business service is used in subsequent task steps, then the Business Rule Service output property set must be parsed.

The process of creating and deploying an entire UI task requires that you complete several tasks such as laying out the task flow, defining task groups, creating task views and binding the views to task steps, defining visibility of the task, testing, and so on.

For a complete example of planning and implementing a UI task that executes a rules module and uses the rule module's output property data as branching criteria, see "Scenario for Using Rules to Provide Dynamic Navigation in a Siebel Task" on page 128.

For information on the Business Rule Service business service, see "About the Business Rule Service Business Service" on page 66. For requirements for setting the Business Rule Service input properties GetMoreData and PerformAction, see "Setting the GetMoreData Input Property" on page 69 and "Setting the PerformAction Input Property" on page 70.

For detailed information on scripting calls to the Business Rule Service business service, see Chapter 10, "Implementing Rules in Scripts."

For detailed information on creating and deploying UI tasks, see *Siebel Business Process Framework: Task UI Guide*.

Scenario for Using Rules to Provide Dynamic Navigation in a Siebel Task

This topic provides an end-to-end example in which a task is configured to create an opportunity, categorize the quality of the lead based on its potential revenue, and branch accordingly to populate other fields on the opportunity. Output from the rules engine provides the criteria for the branching.

Each of the following sections applies to one job role on the Siebel project that is responsible for completing the implementation tasks in that section.

The two job roles are:

- **Business analyst or business developer.** This individual is responsible for specifying the business process flow, the business logic requirements, and the UI requirements.
- **Siebel configuration developer.** This individual implements the requirements provided by the business analyst or business developer by performing configuration tasks in Siebel Tools, HaleyAuthority, and in the Siebel application.

The implementation process includes the following subprocesses and top-level tasks, by role.

- **Design process.** The tasks in this process are done by the business analyst or business developer:

1. "Defining the Business Process Flow" on page 129
2. "Creating View Mock-ups" on page 129
3. "Specifying Business Logic Requirements" on page 133
4. "Specifying Invocation Requirements" on page 134

- Development process. The tasks in this process are done by the Siebel configuration developer:

5. "Creating the Task" on page 135
6. "Creating Business Component and Task Group Definitions" on page 136
7. "Creating UI Elements" on page 137
8. "Binding Views to Task Steps" on page 145
9. "Adding Business Logic" on page 146
10. "Deploying and Activating Rule Modules" on page 152
11. "Testing Rule Modules" on page 153
12. "Providing Calls to the Rules Engine" on page 155
13. "Deploying, Administering, and Testing the Task" on page 161

Defining the Business Process Flow

This topic is a task in the design process that is listed in "Scenario for Using Rules to Provide Dynamic Navigation in a Siebel Task" on page 128.

This task can be done in any application that supports flowcharting or other representation of process flow.

The flowchart in Figure 3 represents the business process for which the task will be created.

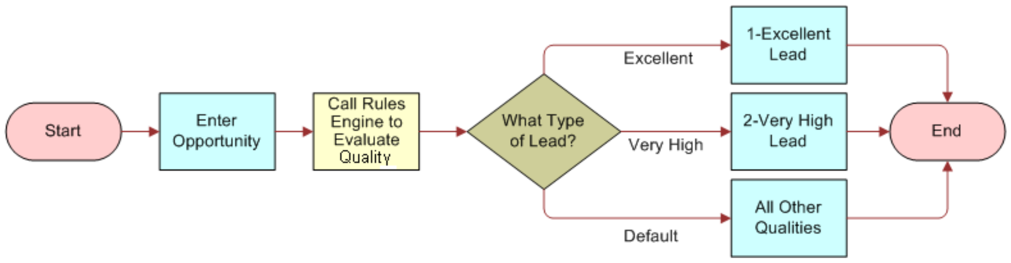


Figure 3. New Lead Business Process Flow

The business process is:

1. Create a new opportunity.
2. Enter fields, including Revenue.
3. Populate the Quality field of the opportunity depending on the amount of potential revenue.
4. Depending on the quality of the lead, branch to a view that requests the user to enter a particular comment field.

This task is based on the Opportunity business object and the Opportunity business component.

Creating View Mock-ups

This topic is a task in the design process that is listed in "Scenario for Using Rules to Provide Dynamic Navigation in a Siebel Task" on page 128.

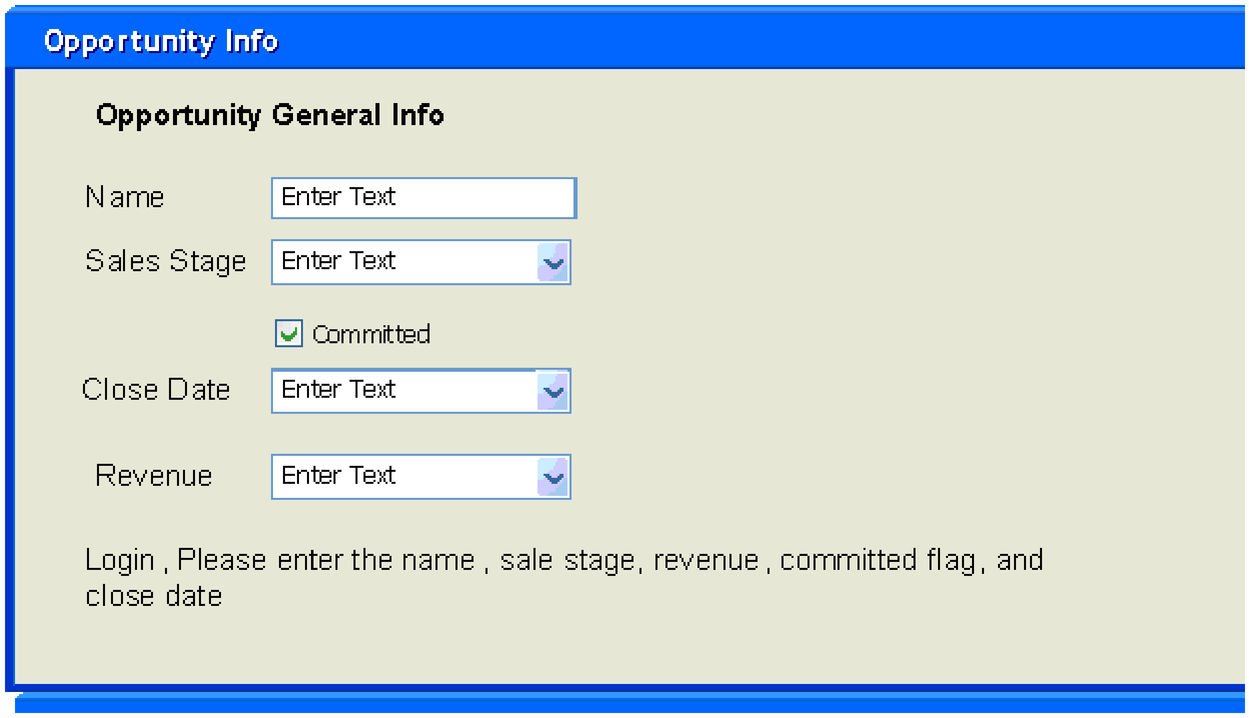
Four views are required for this business process flow:

- **Opportunity General Info view.** User enters data for the new opportunity.
- **Excellent Lead Info view.** User enters data for an excellent lead.
- **Very High Lead Info view.** User enters data for a very high lead.
- **Default Lead Info view.** User enters data for a lead that is neither excellent nor very high.

The following topics provide requirements for each view.

Opportunity General Info View Requirements

Figure 4 is a mock-up of the Opportunity General Info View. It is followed by UI element requirements for the view.



Login , Please enter the name , sale stage, revenue , committed flag, and close date

Figure 4. Opportunity General Info View Mock-up

Fields

Field Name	Type
Name	String
Sales Stage	String
Committed	Char
Close Date	Date
Revenue	Currency

Message The message substitutes the login name of the user, and asks the user to update the fields displayed.

Buttons The Previous button is disabled because this is the first view. All other buttons are enabled.

Excellent Lead Info View Requirements

Figure 5 is a mock-up of the Excellent Lead Info View. It is followed by UI element requirements for the view.

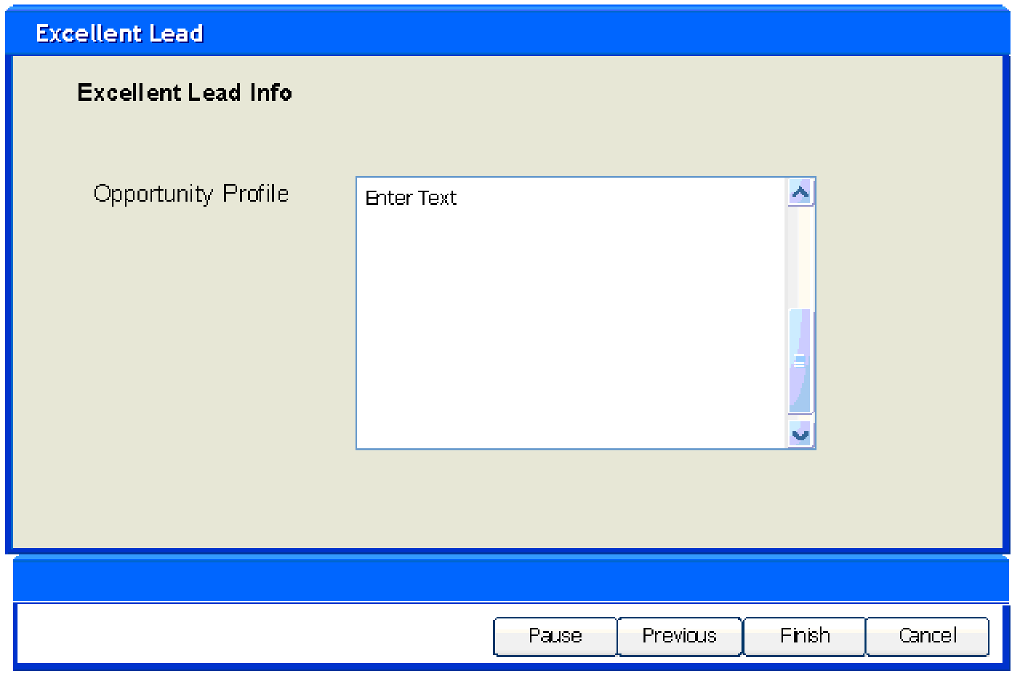


Figure 5. Excellent Lead Info View Mock-up

Fields

Field Name	Type
Opportunity Profile	String

Buttons All other buttons are enabled. Finish Button is exposed.

Very High Lead Info View Requirements

Figure 6 is a mock-up of the Very High Lead Info View. It is followed by UI element requirements for the view.

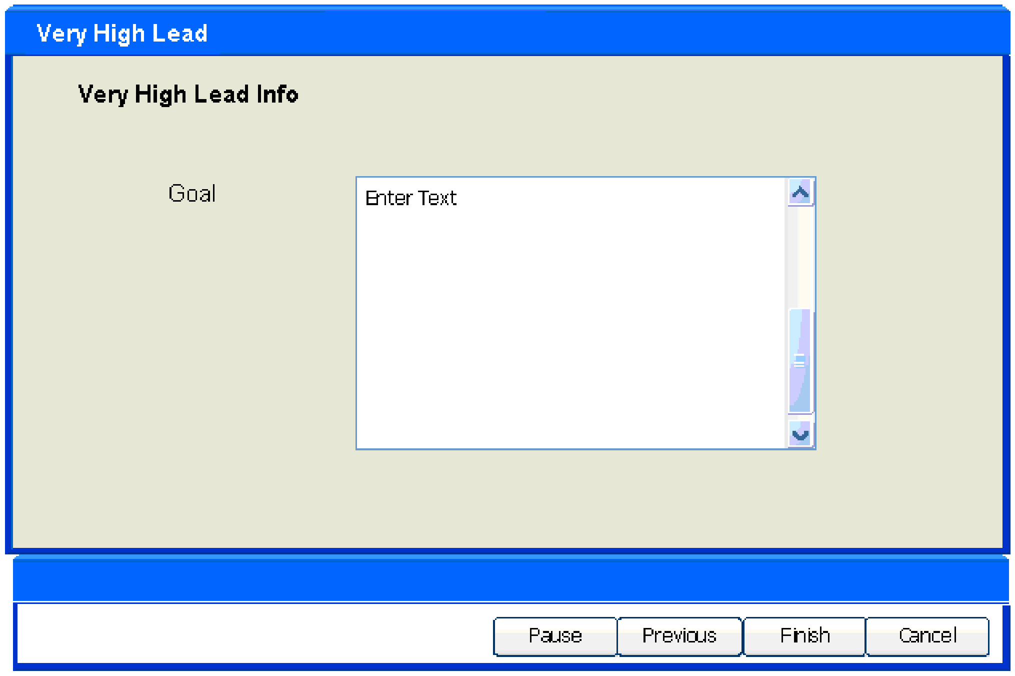


Figure 6. Very High Lead Info View Mock-up

Fields

Field Name	Type
Goal	String

Buttons All other buttons are enabled. Finish Button is exposed.

Default Lead Info View Requirements

Figure 7 is a mock-up of the Default Lead Info View. It is followed by UI element requirements for the view.

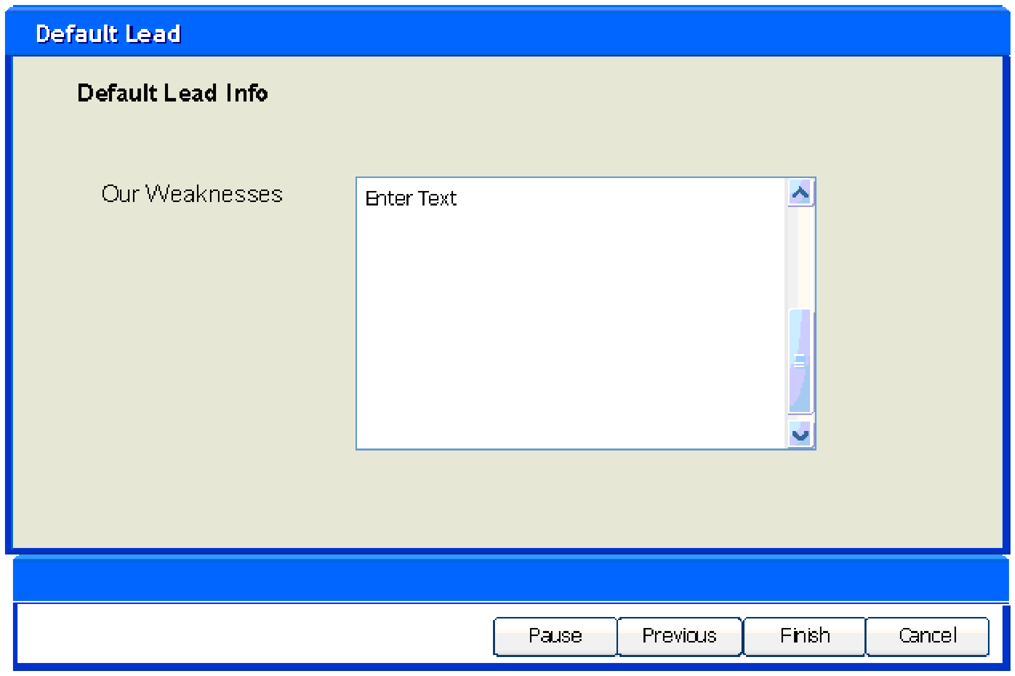


Figure 7. Default Lead Info View Mock-up

Fields

Field Name	Type
Our Weaknesses	String

Buttons All other buttons are enabled. Finish Button is exposed.

Specifying Business Logic Requirements

This topic is a task in the design process that is listed in "Scenario for Using Rules to Provide Dynamic Navigation in a Siebel Task" on page 128.

After the user enters opportunity information, the quality of the lead must be evaluated based on revenue amount, and the lead quality is then used to determine the next view that the user will process.

The quality of the lead is determined as follows:

Lead Quality	Revenue
Excellent	$2,000,000 and greater
Very High	At least $500,000 and less than $2,000,000
Default	At least $1, and less than $500,000

The rules engine looks at the revenue entered and determines the lead quality. Based on the lead quality, one of the following branches is taken:

- **Excellent.** The value of the Quality field on the opportunity is set to 1 - Excellent, and the user is routed to the Excellent Lead Info view to complete the Opportunity Profile field.
- **Very High.** The value of the Quality field on the opportunity is set to 2 - Very High, and the user is routed to the Very High Lead Info view to complete the Goal field.
- **Default.** The value of the Quality field on the opportunity is set to 3 - High, and the user is routed to the Default Lead Info view to complete the Weaknesses field.

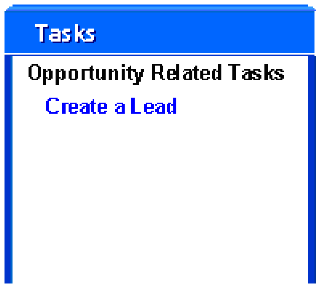
Specifying Invocation Requirements

This topic is a task in the design process that is listed in "Scenario for Using Rules to Provide Dynamic Navigation in a Siebel Task" on page 128.

A user must be able to invoke this task from these views in the Siebel application:

- Opportunity List View
- Manager's Opportunity List View
- All Opportunity List View
- All Opportunities across My Organizations
- All Opportunities across Organizations

The user can drill into the Create a Lead task in the Task pane to invoke the task.



There are no chapters (subtasks) required.

Creating the Task

This topic is a task in the development process that is listed in "Scenario for Using Rules to Provide Dynamic Navigation in a Siebel Task" on page 128.

To allow you to assign display names to Task objects, confirm that the following parameter is set in the [Siebel] section of \TOOLS_ROOT\\bin\LANGUAGE\\tools.cfg:

```
EnableToolsConstrain = FALSE
```

If either parameter is not present, add it.

In Siebel Tools, navigate to Project in the Object Explorer. Create a new project with which to associate all of the metadata artifacts. Name the project, for example Rules Call Back, and lock the project.

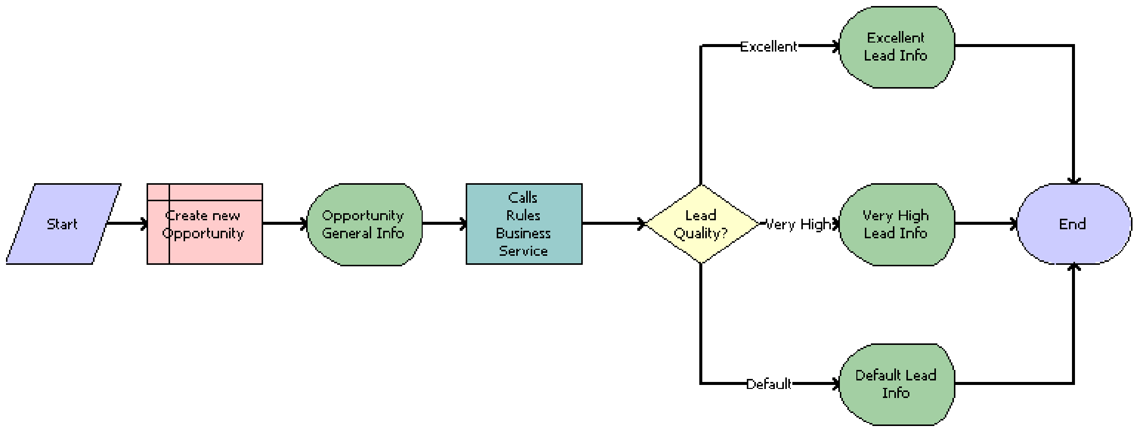
To create the Create a Lead task

1 In the Siebel Tools Object Explorer select the Task Object. Right click and choose New Task Wizard. In the New Task dialog box, enter the following values and click Finish.

Field	Value
Project	Rules Call Back
Task name	Create a Lead
Display name	Create a Lead
Business Object	Opportunity
Default Transient Business Component	Leave empty
Create as a subtask	Unchecked

2 In the Task Editor, model a task similar to the following.

Note that you are simply drawing a picture at this point. You configure the metadata in subsequent steps.



Enter names for steps and labels for connectors as shown.

- Click on a step and enter the name of a step in the Name field of the Properties window for the step.
- Click on a connector between steps and enter the label for the connector in the Label field of the Properties window for the connector.

Creating Business Component and Task Group Definitions

This topic is a task in the development process that is listed in "Scenario for Using Rules to Provide Dynamic Navigation in a Siebel Task" on page 128.

No new Transient Business Components are required for this task, and the task leverages the Opportunity business object and the Opportunity business component.

You must create a task group to define the views in which the Create a Lead task is available.

To create the task group to support invoking this task from the required opportunity-based views

1 Navigate to the Task Group object in the Object Explorer.

If the Task Group object is not displayed, then choose View > Options > Object Explorer and select Task Group.

2 Create a new Task Group record with the following properties:

   Name = Lead Tasks

   Project = Rules Call Back

   Display Name String Override = Lead Tasks

3 Expand the Task Group object in the Object Explorer and select Task Group Item.

4 Create a new Task Group Item record with the following field values in the new task group:

   Action Invoked = Create a Lead

   Type = Task

   Sequence = 1

5 Navigate to the View object in the Object Explorer. Expand the View object to display View Task Group.

   If the View Task Group object is not displayed, then choose View > Options > Object Explorer and select View Task Group.

6 For each of the following views, add Lead Tasks as a view task group with Sequence = 1.

   You may first have to lock each view object or lock its project.

   - Opportunity List View
   - Manager's Opportunity List View
   - All Opportunity List View
   - All Opportunities across My Organizations
   - All Opportunities across Organizations

7 Compile all locked projects and locked objects.

Creating UI Elements

This topic is a task in the development process that is listed in "Scenario for Using Rules to Provide Dynamic Navigation in a Siebel Task" on page 128.

For each of the four views in your task, you define the applet and the view.

To create the applet for the Opportunity General Info view

1 In Siebel Tools, choose File > New Object, then click the Applets tab.

2 Click Form Applet, then click OK to invoke the Form Applet Wizard. In the General dialog box, create a form applet with the following settings, then click Next.

Field	Value
Project	Rules Call Back
Applet name	Opportunity General Info
Applet display title	Opportunity General Info
Business Component	Opportunity
Upgrade behavior	Admin
Use Grid Layout	Checked

3 In the Web Layout - Form dialog box select Edit Mode and click Next.

4 Choose the following fields to add to your applet, then click Next.

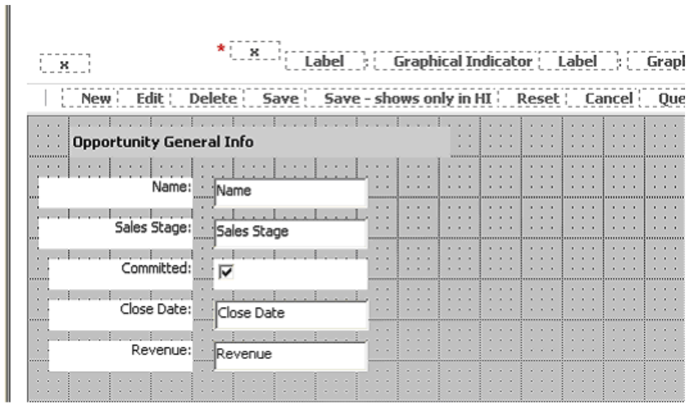
- Name
- Committed
- Sales Stage
- Close Date
- Revenue

5 Choose Next to decline adding controls to your applet, then click Finish.

The Web Template Layout Editor opens.

6 Layout the Opportunity General Info applet to be similar to the following layout.

NOTE: You can add the Opportunity General Info title as a Form Section.



For information about using the Web Layout Editor, see *Using Siebel Tools*.

7 In the Object Explorer, expand the Applet object to display the Applet Message object.

 If the Applet Message object is not displayed, then choose View > Options > Object Explorer and select Applet Message.

8 Add a salutation message to the applet by adding a new Applet Message record with the following properties:

 Name = Salutation Message

 Text Message - String Override = %1, please enter the Name, Sales Stage, Committed Flag, Close Date, and Revenue.

9 In the Object Explorer, expand the Applet Message object to display the Applet Message Variable object.

 If the Applet Message Variable object is not displayed, then choose View > Options > Object Explorer and select Applet Message Variable.

10 Define the variable in your applet message by adding a new Applet Message Variable record with the following properties:

 Field = Updated By Login

 Value = 1

11 Return to the Web Layout Editor and add the new Salutation Message to the applet by dragging a Text control to the canvas. In the Properties window, set the following properties for the control. Unless noted, leave other properties as their default values.

 - Field Type = Message
 - Field = Salutation Message
 - HTML Type = Plain text
 - HTML Only = FALSE
 - Name = Salutation Message (or other descriptive name)

12 Choose File > Save.

You now create a task view that includes the Opportunity General Info applet.

To create the task view that uses the Opportunity General Info applet:

1 In Siebel Tools, choose File > New Object, then click the Task tab.

2 Click Task View, then click OK to invoke the Task View Wizard. In the New View dialog box, create a task view with following settings, then click Next.

Field	Value
Project	Rules Call Back
View name	Opportunity General Info
View title	Opportunity General Info

Field	Value
Business Object	Opportunity
Upgrade behavior	Admin

3. In the View Web Layout - Select Template dialog box, select View Detail 2 (Parent With Pointer) and click Next.
4. In the Web Layout - Applets dialog box, select Opportunity General Info and click Next.
5. In the Task View - Pick Tasks dialog box, when asked if you wish to specify a task applet, select No and click Next.
6. In the Task View - Select Playbar Applet dialog box, select Task Playbar Applet - Bottom, then click the appropriate right arrow to populate the bottom playbar applet text box. Click Next.

   NOTE: Choose the Forward Button type Submit, instead of Next. Submit commits the Opportunity record to the database before the subsequent call of Business Rule Service. Because Business Rule Service is then assured of acting on existing data only, the GetMoreData and PerformAction input properties of Business Rule Service can be set to *Y.* as you can see in "Providing Calls to the Rules Engine" on page 155.

   For requirements for setting the GetMoreData and PerformAction input properties, see "Setting the GetMoreData Input Property" on page 69 and "Setting the PerformAction Input Property" on page 70.

7. In the Finish dialog box, confirm settings and click Finish.

   The Web Template Layout Editor opens, showing the layout of this task view.

Create the following three views by repeating the procedures "To create the applet for the Opportunity General Info view" on page 137 and "To create the task view that uses the Opportunity General Info applet:" on page 139 with the following data:

- **Excellent Lead Info view.** The view you create should be similar to the view shown in Figure 8 on page 141. Use the values in Table 10 on page 141 in your procedures.

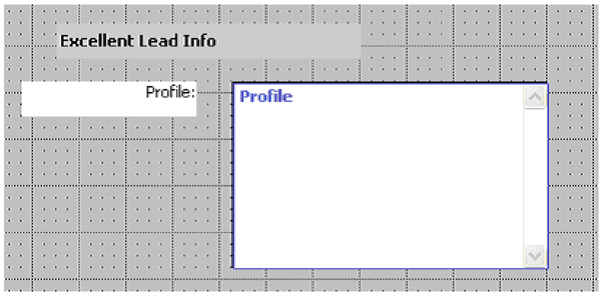


Figure 8. Excellent Lead Info view

Table 10. Data for Excellent Lead Info view

Dialog box or editor	Parameter	Value or Action
Form Applet Wizard	Project	Rules Call Back
	Applet name	Opportunity Excellent Lead
	Applet display title	Opportunity Excellent Lead
	Applet business component	Opportunity
	Applet upgrade behavior	Admin
	Use grid layout	Yes (checked)
Web Layout - Form dialog box	Mode	Edit Mode
Web Layout - Fields dialog box	Add field	Profile
Web Template Layout Editor		Form applet should resemble the applet shown in Figure 8 on page 141
Task View Wizard	Project	Rules Call Back
	View name	Excellent Lead Info
	View title	Excellent Lead Info
	View business object	Opportunity
	View upgrade behavior	Admin
View Web Layout - Select Template dialog box	View template	View Detail 2 (Parent With Pointer)
Web Layout - Applets dialog box	Applet	Opportunity Excellent Lead applet

Table 10. Data for Excellent Lead Info view

Dialog box or editor	Parameter	Value or Action
Task View - Pick Tasks dialog box	Task applet	No
Task View - Select Playbar Applet dialog box	Playbar applet	Task Playbar Applet - Bottom, placed at bottom position

- **Very High Lead Info view.** The view you create should be similar to the view shown in Figure 9 on page 142. Use the values in Table 11 on page 142 in your procedures.

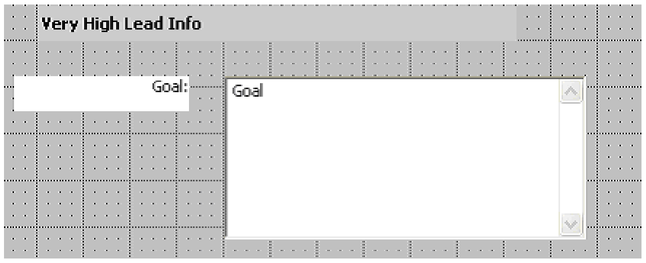


Figure 9. Very High Lead Info view

Table 11. Data for Very High Lead Info view

Dialog box or editor	Parameter	Value or Action
Form Applet Wizard	Project	Rules Call Back
	Applet name	Opportunity Very High Lead
	Applet display title	Opportunity Very High Lead
	Applet business component	Opportunity
	Applet upgrade behavior	Admin
	Use grid layout	Yes (checked)
Web Layout - Form dialog box	Mode	Edit Mode
Web Layout - Fields dialog box	Add field	Goal
Web Template Layout Editor		Form applet should resemble the applet shown in Figure 9 on page 142

Table 11. Data for Very High Lead Info view

Dialog box or editor	Parameter	Value or Action
Task View Wizard	Project	Rules Call Back
	View name	Very High Lead Info
	View title	Very High Lead Info
	View business object	Opportunity
	View upgrade behavior	Admin
View Web Layout - Select Template dialog box	View template	View Detail 2 (Parent With Pointer)
Web Layout - Applets dialog box	Applet	Opportunity Very High Lead applet
Task View - Pick Tasks dialog box	Task applet	No
Task View - Select Playbar Applet dialog box	Playbar applet	Task Playbar Applet - Bottom, placed at bottom position

- **Default Lead Info view.** The view you create should be similar to the view shown in Figure 10 on page 144. Use the values in Table 11 on page 142 in your procedures.

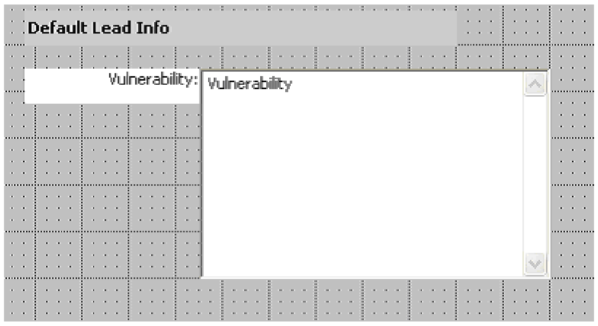


Figure 10. Default Lead Info view

Table 12. Data for Default Lead Info view

Dialog box or editor	Parameter	Value or Action
Form Applet Wizard	Project	Rules Call Back
	Applet name	Opportunity Default Lead
	Applet display title	Opportunity Default Lead
	Applet business component	Opportunity
	Applet upgrade behavior	Admin
	Use grid layout	Yes (checked)
Web Layout - Form dialog box	Mode	Edit Mode
Web Layout - Fields dialog box	Add field	Vulnerability
Web Template Layout Editor		Form applet should resemble the applet shown in Figure 10 on page 144
Task View Wizard	Project	Rules Call Back
	View name	Default Lead Info
	View title	Default Lead Info
	View business object	Opportunity
	View upgrade behavior	Admin
View Web Layout - Select Template dialog box	View template	View Detail 2 (Parent With Pointer)

Table 12. Data for Default Lead Info view

Dialog box or editor	Parameter	Value or Action
Web Layout - Applets dialog box	Applet	Opportunity Default Lead applet
Task View - Pick Tasks dialog box	Task applet	No
Task View - Select Playbar Applet dialog box	Playbar applet	Task Playbar Applet - Bottom, placed at bottom position

Binding Views to Task Steps

This topic is a task in the development process that is listed in "Scenario for Using Rules to Provide Dynamic Navigation in a Siebel Task" on page 128.

You now associate views with the steps in your task.

To bind views to the Create a Lead task

1. In Siebel Tools, navigate to the Task object and select the Create a Lead task.
2. Right-click on the Create a Lead task and choose Edit task flow.
3. In the Task Editor, select and right-click the Siebel Operation step named Create new opportunity, then choose View Properties Window. Assign the following property values in the Properties window. Other values are already set or are read-only.

Property	Value
Business Component	Opportunity
Defer Write Record	TRUE
Operation	Insert
Repeatable	TRUE
Retain Task Search	TRUE

4. Right-click the Task View step that is connected to and follows the Create new opportunity step, then choose Bind Task View.
5. Select the Opportunity General Info view and click OK.

6 Set the following property values for the Opportunity General Info step in the Properties window:

Property	Value
Display Name - String Override	Opportunity General Information
Forward Button Type	Next
Disable Previous	TRUE

7 Repeat Step 4 through Step 6 to bind a view to each of the following Task View steps:

Task View	View to Bind	Display Name - String Override	Forward Button Type	Disable Previous
Task view that is the destination of the branch labeled Excellent	Excellent Lead Info view	Excellent Lead Information	Submit	FALSE
Task view that is the destination of the branch labeled Very High	Very High Lead Info view	Very High Lead Information	Submit	FALSE
Task view that is the destination of the branch labeled Default	Default Lead Info view	Default Lead Information	Submit	FALSE

Adding Business Logic

This topic is a task in the development process that is listed in "Scenario for Using Rules to Provide Dynamic Navigation in a Siebel Task" on page 128.

The business logic requirement is implemented in a business service step that makes a call to the Business Rule Service business service. Business Rule Service calls a rules module that:

- Determines whether a lead is Excellent, Very High, or Default. This return value is assigned to a task property that is used in the decision branching to guide the user along the applicable path of the task flow.
- Sets the quality field for the opportunity to 1-Excellent, 2-Very High, or 3-Default.

You must do the following tasks to add the business logic. Except as noted, these tasks are typical for creating and deploying rules from HaleyAuthority.

- "Creating a Knowledge Base" on page 147. This task is only done once for each environment.
- "Importing Siebel Objects" on page 147, including establishing connectivity to the Master Repository and runtime database on the first import.
- "Organizing Business Logic Into Rules Modules and Submodules" on page 150.

- "Deploying and Activating Rule Modules" on page 152.

Creating a Knowledge Base

For this example, assume this is the first knowledge base created for this application.

To create the knowledge base

1. Launch HaleyAuthority.
   - By default, launch Siebel Business Rules Developer from the Windows program group that contains Siebel Tools.
2. In the HaleyAuthority dialog box, select New knowledge base file (*.akb).

   NOTE: For this example, you create an Access knowledge base and connect to a Siebel sample database. For information on using a non-Access database, see "Setting Up Your Knowledge Base" on page 38.
3. Name the knowledge base, for example TestCallBack, and click Save.

   HaleyAuthority opens with the knowledge base name in the title bar.

Importing Siebel Objects

You import Siebel object definitions - business objects, business components, and fields - from the Siebel Master Repository in order to create rules modules that reflect the business logic. The business logic for this example is based on the Opportunity business object and Opportunity business component. You also based the Create a Lead task on the Opportunity business object.

On your first import of Siebel Objects into the knowledge base, you provide connectivity parameters to the Master Repository and runtime database.

To import Siebel objects and define connectivity

1. In HaleyAuthority, choose File > Import > Siebel Object.

   Siebel Object Importer Wizard launches.
2. From the Siebel Object Importer Wizard Welcome screen, click Next.
3. In the Login screen, enter database connection parameters, then click Next. For this example, the sample database (sse_samp.dbf) is used for both the repository database and as the runtime database.
   - Specify (or browse to) the Siebel application cfg file (for example, uagent.cfg or siebel.cfg) which points to the applicable database. It is typically located in the \BIN*LANG* directory in your Siebel Mobile Web client root directory, where *LANG* is the language code (for example, ENU).
   - Enter the login credentials to the applicable database.
   - Choose Sample for the datasource.

- For the runtime data connection, choose Local mode.

4 In the Pick Task screen, click the Import Siebel objects radio button, then click Next.

5 From the Pick Business Object screen, select the Opportunity business object from the picklist, then click Next.

6 In the Pick Business Components screen, click the checkbox for the Opportunity business component, then click Next.

7 The Pick Business Component Fields dialog box lists available fields for the Opportunity business component. In the Available Fields list, click to select the Name, Revenue, and Quality fields. Click the right arrow to move the selected fields to the Selected Fields list, then click Next.

8 The Finish screen displays the hierarchy of the business object, business component, and fields to be imported. Confirm to import this hierarchy by clicking Finish.

9 Click No when prompted whether to perform another task.

The Output screen displays a log of the concepts, relations, and dictionary objects that are automatically created in HaleyAuthority.

For more detailed information about importing Siebel objects, see "Using Siebel Object Importer" on page 47.

If you expand the concepts hierarchy, you see phrasings that you can use with HaleyAuthority's natural English language editor to reference the imported business components and their fields with your rule statements. These phrasings are generated automatically by the Object Importer and represent the relations between the business components that are imported and their fields that are imported. The phrasings are all of the form a *business component name* has a *field name*; for example, *an opportunity has a revenue*.

How the Multi-Value Revenue Field is Imported

Revenue is a multi-value field (MVF) on the Opportunity business component with a primary value. Revenue field values are derived from a link between the Opportunity and Revenue business components as follows:

- The Opportunity's Revenue field derives its values from the child Revenue business component's Revenue field for the records in the link.
- One of the values in the Opportunity's Revenue field is the primary revenue. It is defined by the Primary Revenue Id field on the Opportunity business component.

Like other multi-value fields with a primary record, the Object Importer recognizes the link and creates the concepts and relations as described in Table 13.

CAUTION: For this example, three of the Siebel objects that take part in this import have the same name, which is not a typical scenario. Table 13 attempts to explain their roles clearly.

Table 13. Concepts and Relations Created by Object Importer for an MVF with a Primary

Concept or Relation	Concept or Relation Created in this Example	Explanation
concept: *business_component*	opportunity	
concept: *child_business_component*	revenue	This is the Revenue child business component, and not the Revenue field on Opportunity.
concept: *child_business_component _destination_ field*	revenue	This is the Revenue field on the Revenue child business component, and not the Revenue field on the Opportunity business component. **NOTE:** There is only one *revenue* entity after the import, although it is created to represent two different objects. However, it contains all the relations applicable to both contexts. Notice that the import log indicates two lines stating that the *revenue* concept is created.
concept: primary *child_business _component*	primary revenue	This concept is created as a child concept to the concept representing the child business component in the link. That is, primary *child_business _component* is a type of *child_business_component*.
relation: a *business_component* has a *child_business _component*	an opportunity has a revenue	Again, note that *revenue* in this relation is the Revenue child business component in the link, not the Revenue field on Opportunity.

Table 13. Concepts and Relations Created by Object Importer for an MVF with a Primary

Concept or Relation	Concept or Relation Created in this Example	Explanation
relation: a *child_business _component* has a *child_business_component _destination_ field*	`a revenue has a revenue`	The Revenue child business component has a Revenue field.
relation: a *business_component* has a primary *child_business _component*	`an opportunity has a primary revenue`	Again, note that revenue in this relation is the Revenue child business component in the link, not the Revenue field on Opportunity.

When you write a HaleyAuthority rule statement that refers to the value of a multi-value field, in this example `an opportunity's revenue`, you instead use the following construct, using the concepts and relations that are explained in Table 13. The construct essentially provides the path to the primary value:

a *business_component*'s primary *child_business_component*'s *child_business_component_destination_field*

For this example, you would use:

```
an opportunity's primary revenue's revenue
```

Organizing Business Logic Into Rules Modules and Submodules

Your business logic may have natural groupings of rules. The groupings may organize your logic sequentially, by applicability, or for some other purpose.

For this example, the business logic is naturally grouped by the three qualities that a lead may qualify for. The business logic to render as rule modules is the evaluation of the quality of a lead.

A module is a grouping of rule statements. A submodule is a grouping of statements within a module.

To create modules and submodules

1 In HaleyAuthority, navigate to the Module Explorer by clicking its tab at the bottom of the screen.

2 Select Modules & Statements, then choose Object > Add a module, and name the new module New Lead.

3 With the New Lead module selected, choose Object > Add a submodule, and name the new submodule Excellent.

4 Repeat Step 3 to create two more submodules to the New Lead module. Name them Very High and Default.

Now you add statements to each submodule to execute your business logic.

To add statements to submodules

1. Select the Excellent submodule, and choose Object > Add a statement.
2. In the Edit Statement dialog box, type in the following statement or pick its components (except quoted strings, which you must type in) from the Words picklist, then click OK. Do not capitalize any words, except those that are in quoted strings.

   ```
   if an opportunity's primary revenue's revenue is at least 2000000 in "USD" then set
   "Quality" of the opportunity to "1-Excellent" and add "Excellent" for the opportunity
   ```

3. Verify that your statement is understood by HaleyAuthority. If it is not understood, a question mark is displayed at the beginning of the statement when it is displayed in the Module Explorer. If your statement is not understood, choose Object > Reparse statement. If the statement is still not understood, choose Object > Edit, and make changes so that the statement matches the direction in this procedure.

   The statement you added includes the following components:

Statement Component	Purpose
`if`	Provides conditional context
`an opportunity's`	The current opportunity
`primary revenue's revenue`	The primary revenue amount For information on the concepts and relations that are created by Object Importer for a multi-value field with a primary, and an explanation of its application to this example, see "How the Multi-Value Revenue Field is Imported" on page 148.
`is at least 2000000 in "USD"`	Operator evaluates the revenue as an integer literal and its currency code. **NOTE:** A currency amount must be entered in the form shown, with its currency type included in quotes.
`then`	Establishes the consequent for the conditional statement.
`set "Quality" of the opportunity to "1-Excellent"`	Sets the Quality field on the Opportunity business component's record to the value *1-Excellent*. **NOTE:** The Quality field is based on a list of values (LOV), so the value must be an exact match to evaluate to TRUE.
`and`	Appends another action.
`add "Excellent" for the opportunity`	Adds *Excellent* as a Derivation value to the output property set that is returned from the rule call if this rule applies and is executed.

4 Repeat Step 1 through Step 3 to add the following statement to the Very High submodule:

```
if an opportunity's primary revenue's revenue is at least 500000 in "USD" and an
opportunity's primary revenue's revenue is less than 2000000 in "USD" then set
"Quality" of the opportunity to "2-Very High" and add "Very High" for the opportunity
```

5 Repeat Step 1 through Step 3 to add the following statement to the Default submodule:

```
if an opportunity's primary revenue's revenue is at least 1 in "USD" and an
opportunity's primary revenue's revenue is less than 500000 in "USD" then set
"Quality" of the opportunity to "3-High" and add "Default" for the opportunity
```

The rules create the following matrix:

Lead Quality	Revenue
Excellent	$2,000,000 and greater
Very High	At least $500,000 and less than $2,000,000
Default	At least $1 and less than $500,000

This example does not illustrate testing the rule module with the HaleyAuthority test harness before deploying the rule module. Instead, see "Testing Rule Modules" on page 153 for an alternate testing strategy for the rule module and the Business Rule Service business service. For information and examples of using the test harness, see "Using the HaleyAuthority Test Harness" on page 56, "Scenario for Using Rules to Validate Data at Runtime" on page 78, or "Scenario for Using Rules in a Siebel Workflow to Do Batch Processing" on page 104.

Deploying and Activating Rule Modules

This topic is a task in the development process that is listed in "Scenario for Using Rules to Provide Dynamic Navigation in a Siebel Task" on page 128.

You must deploy the New Lead module to a runtime environment.

To deploy the rule module

1 In HaleyAuthority, choose Tools > Siebel Deployment.

2 In the Siebel Deployment dialog box, select the New Lead from the List of Modules list, check the Update model checkbox, then click OK.

   This step logs you into the runtime database and deploys the New Lead rules definitions to the runtime tables.

To activate the rule module

1 Log into the Siebel application in the same runtime environment to which you deployed the rule module.

2 Navigate to Business Rules Administration > Rule Module List view.

In the Rules Modules applet, New Lead is listed as a module.

3 Set the Business Object of the New Lead module to Opportunity.

4 In the Rule Module Relation applet, create three new records and assign the business components as follows:T

Parent Business Component	Business Component
No entry	Opportunity
Opportunity	Primary Revenue
Opportunity	Revenue

5 In the Rules Modules applet, click Activate.

Testing Rule Modules

This topic is a task in the development process that is listed in "Scenario for Using Rules to Provide Dynamic Navigation in a Siebel Task" on page 128.

Test the rule module to confirm that it gives the expected results.

The following test uses the Business Service Administration view to test against data in an existing opportunity in the database. To confirm that the rule module assigns the correct type of quality to the opportunity, the test uses an XML file to supply the opportunity with different revenue amounts to the Business Rules Service, the prepackaged business service for making calls to the rules engine.

To test the New Lead rule module

1 Log into the Siebel application in the same runtime environment to which you deployed the rule module.

2 Find an existing opportunity and choose Help > About Record, then its Row Id. Also, note the revenue for this opportunity.

3 Create an XML document with content similar to the following that includes the correct data structure to supply to the business rules service. Substitute applicable URLs and paths and the id and the revenue amount that you recorded.

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

<!--Sample XML file generated by XMLSpy v2005 rel. 3 U (http://www.altova.com)-->

<PropertySet  xmlns="http://www.siebel.com/xml/BizRule" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.siebel.com/xml/
BizRule D:\RulesEngine\Design\Hemingway\Runtime\InputPropertySet.xsd"
RuleModuleName="New Lead" GetMoreData="Y" PerformAction="Y">

   <BusCompList>
```

```
            <BusComp Name="Opportunity" Id="6-48ZVH">
            </BusComp>
         </BusCompList>
      </PropertySet>
```

4 Save the file as, for example, NewLead_Input.xml where you can find it easily.

5 If necessary, log back into the Siebel application and navigate to the Business Service Administration Screen > Business Service Test View.

6 Create a new parent record and assign the following values:

   Service Name = Business Rule Service

   Method = RunRules

   Iterations = 1

7 In the Input Arguments applet, click the Load From File button, select the file created in Step 3, and choose the Run on One Input button from the Simulator applet.

   An output argument record is created.

8 Select the output argument record and click the Save to File button. When you are prompted to save the file, choose Open to show the results from the rule invocation.

   The correct evaluation should be made, depending on the revenue amount in your chosen Opportunity record. The opportunity must evaluate as Excellent, Very High, or Default, and the Quality field must be set to 1-Excellent, 2-Very High, or 3-High, respectively.

   Your output should have a similar structure to the following XML example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<?Siebel-Property-Set EscapeNames="true"?>
<PropertySet>
  <DerivationList>
    <Derivation BusCompId="6-48ZVH" BusCompName="Opportunity">
      <Value>Excellent</Value>
    </Derivation>
  </DerivationList>
  <SetFieldValueList>
    <SetFieldValue BusCompId="6-48ZVH" BusCompName="Opportunity">
      <Field Name="Quality">1-Excellent</Field>
    </SetFieldValue>
  </SetFieldValueList>
```

```
</PropertySet>
```

Providing Calls to the Rules Engine

This topic is a task in the development process that is listed in "Scenario for Using Rules to Provide Dynamic Navigation in a Siebel Task" on page 128.

Now that the rules module has passed a simulation test, the last task is to create and use a business service to call the rule module, provide its inputs, and capture its outputs. You must do two configuration tasks:

- Create a business service that calls the rule module.
- Include the business service in the task definition.

To create a business service that calls the rule module

1. In Siebel Tools Object Explorer, select Business Service.
2. Create a new business service with the following properties:

   Name = Rules Business Call - Task Opportunity

   Project = Rules Call Back

   Cache = TRUE

   Display Name - String Override = Rules Business Call - Task Opportunity

   External Use = TRUE
3. Expand the Business Service and add a business service method with the following properties to the Rules Business Call - Task Opportunity business service you created in Step 2:

   Name = Rules

   Display Name - String Override = Rules
4. Expand the Business Service Method object and add the following business service arguments to the Rules business service method you created in Step 3.

Name	Data Type	Type	Storage Type	Display Name - String Override
vRowId	String	Input	Property	vRowId
vQuality	String	Output	Property	vQuality

5. Right-Click on the Rules Business Call - Task Opportunity business service and choose Edit Server Scripts.

6 Select Service_PreInvokeMethod and enter the following script.

CAUTION: All examples of script in this document are written for use with the ST eScript engine. To implement this example, first confirm that you are set to use the ST eScript engine in Siebel Tools. For information on setting Siebel Tools to use the ST eScript engine, see "To enable the ST eScript engine for Siebel Tools" on page 97.

```
function Service_PreInvokeMethod (MethodName, Inputs:PropertySet,
Outputs:PropertySet)

{

   if (MethodName == "Rules")

   {

      var out:chars = Rules(Inputs, Outputs);

      Outputs.SetProperty("vQuality",out);

      return (CancelOperation);

   }

   return (ContinueOperation);

}
```

7 Add the following custom method to the Rules Business Call - Task Opportunity business service:

```
function Rules(Inputs:PropertySet, Outputs:PropertySet)

{

   try

   {

      //Declare a business service variable.

      var svc:Service = TheApplication().GetService("Business Rule Service");


      //Declare the inputs property set variable for Business Rule Service. Declare

      //child and grandchild property sets that build the

      //hierarchical structure of the BusCompList input property.

      var inputs:PropertySet = TheApplication().NewPropertySet();

      var child:PropertySet = TheApplication().NewPropertySet();

      var grandchild:PropertySet = TheApplication().NewPropertySet();


      //Declare the outputs property set variable for Business Rule Service.
```

```
var outputs:PropertySet = TheApplication().NewPropertySet();

//Declare the variable that gets the row id of the new lead from the task.
var RowIdInput:chars = Inputs.GetProperty("vRowId");

//Declare variables to store results.
var valResult:chars;
var TheResult:chars;

//Use methods to build input property set structure.
//Setting RuleModuleName to module you wish to call.
inputs.SetProperty("RuleModuleName", "New Lead");

//Setting PerformAction to "Y" because our rule statements include
//setfieldvalue calls to process.
inputs.SetProperty("PerformAction", "Y");

//Setting GetMoreData to "Y" because you want the rules engine to get all
//the child business components and fields automatically, instead of passing
//them all explicitly.
inputs.SetProperty("GetMoreData", "Y");

//Build the BusCompList > BusComp > Field hierarchy.
child.SetType("BusCompList");
grandchild.SetType("BusComp");
grandchild.SetProperty("Id", RowIdInput);
grandchild.SetProperty("Name", "Opportunity");
child.AddChild(grandchild);
inputs.AddChild(child);
```

```
// Invoke Business Rule Service.
svc.InvokeMethod("RunRules", inputs, outputs);

// Examine the outputs property set.

var nChild = outputs.GetChildCount();
var i = 0;
var resultType;
var errorText;

// If no result, then the derivation passed in this example.
if (nChild == 0)
{
   return (ContinueOperation);
}

// In this example, you only look for RaiseErrorText and DerivationList
// because you only used those actions in the New Lead rule module.
for (i = 0; i < nChild; i++)
{
   resultType = outputs.GetChild(i).GetType();

   //Process RaiseErrorText result
   if (resultType == "RaiseErrorText")
   {
      errorText = outputs.GetChild(i).GetValue();
      return(CancelOperation);
   }

   // Process DerivationList result.
```

```
      if (resultType == "DerivationList")
      {
         var j = 0;
         var msg:PropertySet = TheApplication().NewPropertySet();
         var valList:PropertySet = outputs.GetChild(i);
         var valCount:float = valList.GetChildCount();

         // Look at each Derivation result.
         for (j = 0; j < valCount; j++)
         {
            valResult = valList.GetChild(j).GetProperty("Result");
            if (valResult != "Valid")
            {
               var val:PropertySet = valList.GetChild(j);
               var msgCount = val.GetChildCount();
               var k = 0;
               for (k = 0; k < msgCount; k++)
               {
                  TheResult = val.GetChild(k).GetValue();
               }
               return (TheResult);
            }
         }
      }
   }
}//end try
catch(e)
{
   TheApplication().RaiseErrorText("Error in the Rules function: " +
   e.toString());
```

```
        }
        finally
        {


        }
    }
```

For information and examples of the output property sets, see "Output Property Set Schema for Business Rule Service" on page 188.

8 Navigate back to the Business Services list, select the Rules Business Call - Task Opportunity business service, and choose Tools > Compile Selected Objects.

Finish the task definition to include the new Rules Business Call - Task Opportunity business service.

To include the business service in the task definition

1 In Siebel Tools, navigate to the Task object and select the Create a Lead task.

2 Right-click and choose Edit Task Flow to access the task editor. Confirm that the status property of the Create a Lead task is In Progress.

3 Select the main grid with no steps selected. Right-click in the Multi Value Property Window and choose New Record to add each of the following properties:

Name	Data Type	Access Mode
varRevenue	Number	R/W
varRowId	String	R/W
ReturnFromRulesEngine	String	R/W

4 Select Opportunity General Info Task View Step. In the Multi Value Property Window, click the Output Arguments tab, then choose New Record to add each of the following output arguments:

Property Name	Type	Business Component	Field
varRevenue	Business Component	Opportunity	Revenue
varRowId	Business Component	Opportunity	Id

Setting these output arguments allows inputs from the user to be assigned to task properties that are ultimately provided as inputs to the business service.

5 Select the Call Rules Business Service business service step and right-click to assign the following property values.

Property	Value
Business Service Method	Rules
Business Service Name	Rules Business Call - Task Opportunity

6 With the Call Rules Business Service business service step selected, click the Input Arguments tab in the Multi Value Property Window. Right-click, then choose New Record to add each of the following input arguments.

Input Argument	Type	Property Name
vRevenue	Task Property	varRevenue
vRowId	Task Property	varRowId

7 Similarly, add the following output argument to the Call Rules Business Service business service step.

Property Name	Type	Output Argument
ReturnFromRulesEngine	Output Argument	vQuality

8 Select the condition branch labeled Excellent. Right-click and choose Edit Conditions. Set the parameters as follows, then click Add.

Compare To	Operation	Object	Value
Task Property	All Must Match (Ignore Case)	ReturnFromRulesEngine	Excellent

9 Repeat Step 8 to set the following condition for each of the Very High and Default condition branches:

Condition Branch	Compare To	Operation	Object	Value
Very High	Task Property	All Must Match (Ignore Case)	ReturnFromRulesEngine	Very High
Default	Task Property	All Must Match (Ignore Case)	ReturnFromRulesEngine	Default

Deploying, Administering, and Testing the Task

This topic is a task in the development process that is listed in "Scenario for Using Rules to Provide Dynamic Navigation in a Siebel Task" on page 128.

You must deploy the Create a Lead task, then administer and test the task in the runtime environment.

To deploy the task

1. In Siebel Tools, navigate to the Task object and select the Create a Lead task. Confirm that the status property of the Create a Lead task is In Progress.
2. On the WF/Task Editor toolbar, click Publish (not Publish/Activate).
3. In the Siebel dialog box, click No to choose to validate the task instead of publishing the task.
4. In the Validate dialog box, click Start to generate errors and warnings. Correct errors and revalidate until all errors are resolved.
5. With the Create a Lead task still selected, click Publish (not Publish/Activate) on the WF/Task Editor toolbar, then click Yes in the Siebel dialog box to publish the task.
6. Choose Tools > Compile Projects, and compile the Rules Call Back project to the repository that is used by your sample database (typically <*Siebel client root directory*>\OBJECTS\<*language*>\siebel.srf.

You must activate the published Create a Lead task.

To activate the published task

1. In the Siebel application in your runtime environment, navigate to the Administration - Business Process screen and click the Task Deployment link.
2. In the Published Tasks list, query for the Create a Lead task.
3. Click the Activate button.

Administering the Create a Lead task includes specifying the responsibilities that may run the task.

To administer the task

1. In the Siebel application in your runtime environment, navigate to the Administration - Application screen. Click the Tasks link.
2. In the Registered Tasks list, create a new record.
3. From the Task Name picklist choose the Create a Lead task.
4. With the Create a Lead record selected in the Registered Tasks list, navigate to the Responsibilities list and add all the responsibilities that can run the Create a Lead task.

   NOTE: Be sure to include a responsibility that is included in your position in order for you to run the task.

5. Navigate back to the Registered Tasks list and click the Clear Cache button.

Now you can test the Create a Lead task in the runtime environment.

To test the task in the runtime environment

1. In the Siebel application in your runtime environment, navigate to the Opportunity screen and click the Opportunity List link.
2. In the Opportunity List view, click the Toggle Tasks button in the Application toolbar.

   The Tasks pane displays and should include the Create a Lead task.
3. Click the Create a Lead link.

   The first view in the task should display with the following components:

   - Areas to enter the name, sales stage, committed flag, close date, and revenue for a new opportunity. Note the red asterisks for required fields.
   - The message that addresses you and requests you to populate all the fields exposed on the view.
   - The playbar applet that provides navigation through the task.
4. Enter 5000000 as the revenue and click Next.

   Because $5,000,000 qualifies as an Excellent opportunity, the task should branch to the Excellent Lead Info view.
5. Click Previous, change the revenue to 500000, and click Next.

   Because $500,000 qualifies as a Very High opportunity, the task should branch to the Very High Lead Info view.
6. Click Previous, change the revenue to 50000, and click Next.

   Because $50,000 qualifies as a Default opportunity, the task should branch to the Default Lead Info view.
7. Enter data and continue to finish the task.
8. Navigate to the Opportunity List view and confirm that a new opportunity record is created with the expected field values, including the task-generated value for the Quality field.

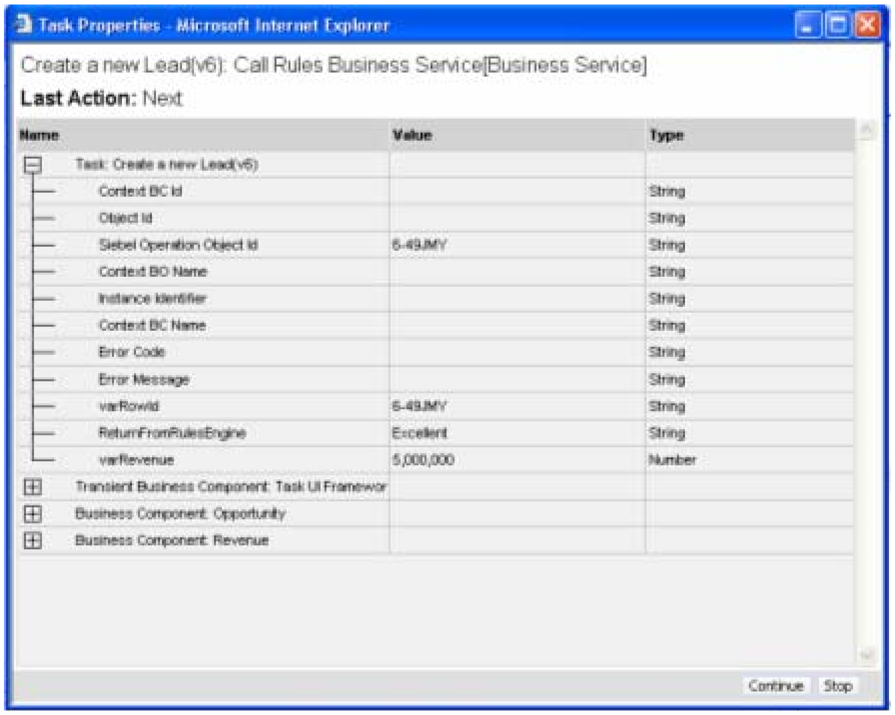
As another level of testing, you can step through the task in debug mode. To enable debug mode, you must set a parameter in the Siebel application's cfg file. For this example, the cfg file is located in the Mobile Web Client.

To run the task in the debug mode

1. Log out of the Siebel application.
2. In a text editor, open the cfg file for the Siebel application; for example \SIEBEL_ROOT\webclient\bin\enu\uagent.cfg for Siebel Call Center.
3. In the [InfraUIFramework] section, set EnableRestrictedMenu = TRUE. If the parameter does not exist, then add it.

4 Log back into the Siebel application and re-invoke the Create a Lead task.

For each step in the task, the Task Properties window displays data for the step such as the following for the Call Rules Business Service step.



Create a new Lead(v6): Call Rules Business Service[Business Service]

Last Action: Next

Name	Value	Type
Task: Create a new Lead(v6)		
Context BC Id		String
Object Id		String
Siebel Operation Object Id	6-49JMY	String
Context BO Name		String
Instance Identifier		String
Context BC Name		String
Error Code		String
Error Message		String
varRowId	6-49JMY	String
ReturnFromRulesEngine	Excellent	String
varRevenue	5,000,000	Number
Transient Business Component: Task UI Framewor		
Business Component: Opportunity		
Business Component: Revenue		

Best Practices for Implementing Rules in Siebel Tasks

Table 14 provides best practices to follow when you implement rules in Siebel Task UI.

Table 14. Best Practices for Implementing Rules in Siebel Task UI

Guideline	Explanation
Independent of how you call the Business Rule Service business service, to capture its output data, you must provide custom script that calls the Business Rule Service business service and parses its output property set.	
To optimize performance: When directly invoking the Business Rule Service business service, such as from scripts or workflow, it is recommended to set the GetMoreData input property to *N* if you can construct the necessary data required for the BusCompList input property.	For a given business component that has been imported into HaleyAuthority, various descendant business components and fields may have been imported into HaleyAuthority. For a given rules module that acts on the business component, it is likely that only a subset of the business component's descendant business components and fields are used by the rule module. Setting GetMoreData to N requires you to explicitly provide the business component-field hierarchy to the BusCompList input property, but it avoids having to process the entire hierarchy of a business component when the rule module executes.
To optimize performance: Minimize the number of invocations to the Business Rule Service business service.	Every rule module invocation incurs a CPU cost.
To optimize performance: It is preferable to implement a large number of rule statements by: ■ Invoking the Business Rule Service business service a lesser number of times with rule modules that contain a greater number of rules, instead of ■ Invoking the Business Rule Service business service a greater number of times with rule modules that contain a lesser number of rules	In general, invocations of the Business Rule Service business service provide a much greater impact to performance than large rule modules.

13 Administering Rules

The specifics of business processes change over time. After creating, deploying, and implementing rules modules to implement particular business processes, you must keep those processes current with changes in personnel, business process requirements, parameters for business processes, changes to business objects, and so on.

This chapter provides instruction on the continuing administration of business rules. The chapter contains the following topics:

- "Keeping the HaleyAuthority Knowledge Base Consistent with the Siebel Repository" on page 167
- "Modifying and Re-Deploying Rules" on page 167

Keeping the HaleyAuthority Knowledge Base Consistent with the Siebel Repository

The representation of Siebel objects as HaleyAuthority concepts and relations must be kept consistent with the Siebel object model. Changes to the Siebel object model require that you synchronize the HaleyAuthority knowledge base with the Siebel Repository.

For information on synchronizing the HaleyAuthority knowledge base with the Siebel Repository, see "Synchronizing the HaleyAuthority Knowledge Base with the Siebel Repository" on page 50.

Modifying and Re-Deploying Rules

As business processes and logic change in your organization, you may need to modify your rules.

For information about reconfiguring and deactivating existing rules, see "Reconfiguring and Deactivating Rule Modules" on page 65.

A Reference Topics for Siebel Business Rules

This appendix contains the following reference topics:

Rules Glossary

Table 15 provides definitions of rules terminology.

Table 15. Rules Terminology

Term	Definition
Action	A procedural relation that satisfies at least one of the following conditions: - Returns no value - Produces output, consumes input, modifies data, or changes state in any other way For example, the *add-value* action is phrased as *add a value for an entity*.
Function	A procedural relation that satisfies both of the following conditions: - Returns values - Does not produce output, consume input, modify data, or change state in any other way For example, the *get-profile-attribute* function is phrased as *a first string is profile attribute of a second string*.

Table 15. Rules Terminology

Term	Definition
knowledge base	**NOTE:** There are two uses of this term, with signficantly different definitions. When *HaleyAuthority* or runtime does not precede *knowledge base*, assume the first definition. HaleyAuthority knowledge base: A database that stores HaleyAuthority schema and data. Runtime knowledge base: A representation of the semantic role model and rules that is instantiated in the Application Object Manager at runtime.
Predicate	A function that returns one value, either True or False
rule module	A top-level rule module is a unit of rule definition, deployment, and execution. A rule module may contain sub modules or rule statements (or simply rules). A sub module may contain rule statements. A rule module is also called agenda at runtime.
semantic role model	In HaleyAuthority, the set of concepts and the relations among them. In order for rules to be written, the objects and relations on which the rules depend must exist as part of the semantic role model.

Siebel Object Model Mapping to the HaleyAuthority Semantic Role Model

Siebel Object Importer generates HaleyAuthority objects and relations to represent the Siebel objects that are imported and their inherent relationships. Table 16 describes how Siebel objects and relationships are mapped to HaleyAuthority objects and relations by Siebel Object Importer.

A HaleyAuthority concept in Table 16 is represented as its position in the concepts hierarchy, where a colon separates the levels in the hierarchy. For example, assume *value: date, specific day* is the HaleyAuthority Concept entry for a Siebel field type. *Value* is a type (or child) of concept, and *date, specific day* is a type of value.

Table 16. Siebel Object Model to HaleyAuthority Mapping

Siebel Object	HaleyAuthority Concept	Explanation
Business Component	entity	
Field of Type DTYPE_BOOL	value: boolean	

Table 16. Siebel Object Model to HaleyAuthority Mapping

Siebel Object	HaleyAuthority Concept	Explanation
Field of Type DTYPE_CLOB		Not mapped, and not available to choose in the Object Importer. The length of CLOB can exceed HaleyAuthority's string limit.
Field of Type DTYPE_CURRENCY	entity: currency value	
Field of Type DTYPE_DATE	time: period of time: specific period of time: date, specific day and value: date, specific day	
Field of Type DTYPE_DATETIME	Not available to choose in the Object Importer	Not mapped, and not available to choose in the Object Importer. DTYPE_DATETIME fields are not UTC time. HaleyAuthority stores all time as UTC time, so these fields cannot be used in predicates.
Field of Type DTYPE_ID	value: string	
Field of Type DTYPE_INTEGER	quantity: number: decimal number: integer and value: number: decimal number: integer	
Field of Type DTYPE_NOTE	value: string	
Field of Type DTYPE_NUMBER	quantity: number: decimal number: real number and value: number: decimal number: real number	
Field of Type DTYPE_PHONE	value: string	
Field of Type DTYPE_TEXT	value: string	

Table 16. Siebel Object Model to HaleyAuthority Mapping

Siebel Object	HaleyAuthority Concept	Explanation
Field of Type DTYPE_TIME	time: point in time: recurrent point in time: time of day **NOTE:** A DTYPE_TIME field can be a point in time, like the time for an activity to begin, or an interval of time, like the duration of a phone call. Object Importer maps all fields of this type to the time of day concept.	
Field of Type DTYPE_UTCDATETIME	time: point in time: instant, specific point in time and value: instant, specific point in time	
Calculated Field of Type NULL	Not available to choose in the Object Importer	Object Importer imports all the currencies currently in the Master Repository into HaleyAuthority.
Currency	unit of money	
Multi-value field (MVF)		Instead of modeling a multi value field as a field of its parent business component, it is instead modeled as its destination business component and the corresponding field. For example, in the case of *Primary Bill To Last Name* field of *Account*, the *Contact* business component and its field *Last Name* are automatically imported. Concepts are created for *Contact* business component, primary contact, and *Last Name* field. Relations are created between: *contact* and *last name* *account and contact* *account and primary contact*
Link	one-to-many relation: parent business component, child business component	One-to-Many relation. A many-to-many relation between business components is represented as two links in Siebel. It is mapped to two relations in HaleyAuthority.

Table 16. Siebel Object Model to HaleyAuthority Mapping

Siebel Object	HaleyAuthority Concept	Explanation
Multi-value group (MVG) link (Traditional multi-value link with *Primary Id Field* property specified)	A concept is created for a kind of destination business component (the primary). A one-to-one relation is created: root business component, concept described above.	Map only when the Primary Id property is not null, because the main purpose is to model the Primary field value. For example, for the *Bill To Contact* field of the *Account* business component, the following concepts are created: Concept *primary bill to contact* as a kind of contact entity relation between *account* and *primary bill to contact* Rules Runtime must determine the primary so that it can assert the data correctly.
MVG link (Traditional multi-value link with *Primary Id Field* property unspecified)	No concepts or relations are created.	
MVG link (Indirect multi-value link with *Primary Id Field* property specified)	Relation: parent business component, destination business component A concept is created for a kind of destination business component (the primary). One-to-one relation is created: root business component, concept described above.	In this case, need to treat the indirect multi-value link as a link between this business component and the destination business component. For example, *Account Issue Action* business component has an indirect multi-value link *Contact* whose destination link is *Action\Contact*. The one-to-many relationship between *Account Issue Action* and *Contact* is only kept in this indirect multi-value link, and not in the destination link as with a traditional multi-value link.
MVG link (Indirect multi-value link with *Primary Id Field* unspecified)	Relation: parent business component, destination business component	

Siebel Concepts, Relations, Actions, Functions, and Predicates

Table 17 describes the concepts and relations that are created in HaleyAuthority when Siebel objects are first imported into a knowledge base by Siebel Object Importer.

All currency-related objects are provided because HaleyAuthority does not fully support currency with exchange date and currency code.

Table 17. Siebel Concepts and Relations in HaleyAuthority

HaleyAuthority Object Type	Name	Explanation
entity	currency value	Required for currency operations.
date, specific day	exchange date	Required for currency operations.
string	currency code	Required for currency operations.
relation	currencyvalue_currencycode	Phrasing: *a currency value has a currency code*
relation	currencyvalue_exchangedate	Phrasing: *a currency value has an exchange date*
relation	currencyvalue_realnumber	Phrasing: *a currency value has a real number*

Table 18 describes the actions, functions, and predicates that are created in HaleyAuthority when Siebel objects are first imported into a knowledge base by Siebel Object Importer.

For definitions of action, function, and predicate, see "Rules Glossary" on page 169.

For detailed information about HaleyAuthority terminology, see *HaleyAuthority Help*.

Table 18. Siebel Actions, Functions, and Predicates in HaleyAuthority

Action type	Action	Purpose	Example of Phrasing	Example Requires Importing These Objects:

action	add-currency-literal	For derivation type of rules, add a currency literal as a derived value from rules.	`if an account's location contains "CA" then add 50000 in "USD" for the account`
			
			`if an account's location contains "CA" then add 70000 in "USD" on today for the account`
			
			`if an account's location contains "Germany" then add 700000 in "DEM" on 2006/03/01 for the account`
				Account BO
				Account BC
				Location

action	add-currency-value	For derivation type of rules, add a currency value as a derived value from rules.	`If an account's name is not unknown then add the account's annual revenue for the account`	Account BO Account BC Annual Revenue Name
action	add-value	For derivation type of rules, add a value as a derived value from rules. The value can be any non-currency value types that are supported. For a list of types supported by Object Importer, see Table 16 on page 170.	`If an account's annual revenue is more than 1000000 in "USD" then add "Gold Account" for the account`	Account BO Account BC Annual Revenue

Table 18. Siebel Actions, Functions, and Predicates in HaleyAuthority

Action type	Action	Purpose	Example of Phrasing	Example Requires Importing These Objects:

action	invalidate-with-reason	For validation type of rules, invalidate with reason	`If an account's location is unknown then invalidate the account with "Please provide a location for the account"`
			
			`You can capture multiple rules violations with a single message statement with applicability statements for each type of violation.`
				Account BO
				Account BC
				Location

action	raise-error-text	Raise an error and stop the rules execution	`If an account's name is unknown then raise an error as "Account should have a name"`	Account BO Account BC Name
action	set-currency-field-value	Set a Siebel business component field of type DTYPE_CURRENCY	`Set "Annual Revenue" of an account to the account's revenue`	Account BO Account BC Revenue
action	set-currency-field-value-literal	Set a Siebel business component field of type DTYPE_CURRENCY	`Set "Annual Revenue" of an account to 9000000 in "USD"`	Account BO Account BC
action	set-field-value	Set a Siebel business component field of types other than DTYPE_CURRENCY	`If an account's location contains "US" then set "Description" of the account to "This account is in the US"`	Account BO Account BC Location
action	set-profile-attribute	Set a profile attribute	`Set profile attribute of "hobby" to "fishing"`	Does not depend on any object import.

Table 18. Siebel Actions, Functions, and Predicates in HaleyAuthority

Action type	Action	Purpose	Example of Phrasing	Example Requires Importing These Objects:
action	validate	For validation type of rules, validate an entity	`If an account's name is not unknown and the account's location is not unknown then validate the account`	Account BO Account BC Name Location
function	currency-literal-divided-by-literal	Arithmetic operation with currency values and currency literals	`if 100 in "USD" on 2005/03/01 divided by 100 in "USD" on 2006/03/01 is more than 1 then …`	Does not depend on any object import.
function	currency-literal-divided-by-number	Arithmetic operation with currency values and currency literals	`If 100 in "EUR" divided by 10 is equal to 10 in "EUR" then …`	Does not depend on any object import.
function	currency-literal-divided-by-value	Arithmetic operation with currency values and currency literals	`If 1000000 in "USD" divided by an account's annual revenue is more than 1 then …`	Account BO Account BC Annual Revenue
function	currency-literal-minus-literal	Arithmetic operation with currency values and currency literals	`If 100 in "USD" on today minus 100 in "USD" on yesterday is equal to 0 in "USD" then …`	Does not depend on any object import.
function	currency-literal-minus-value	Arithmetic operation with currency values and currency literals	`If 1000000 in "USD" minus an account's annual revenue is less than 0 in "USD" then …`	Account BO Account BC Annual Revenue
function	currency-literal-plus-literal	Arithmetic operation with currency values and currency literals	`If 100 in "USD" plus 100 in "KRW" is more than 200 in "USD" then …`	Does not depend on any object import.

Table 18. Siebel Actions, Functions, and Predicates in HaleyAuthority

Action type	Action	Purpose	Example of Phrasing	Example Requires Importing These Objects:
function	currency-literal-plus-value	Arithmetic operation with currency values and currency literals	`If 1000000 in "USD" plus an account's annual revenue is more than the account's revenue then …`	Account BO Account BC Annual Revenue Revenue
function	currency-literal-times-number	Arithmetic operation with currency values and currency literals	`If 100000 in "USD" times 12 is less than an account's annual revenue then`	Account BO Account BC Annual Revenue
function	currency-value-divided-by-literal	Arithmetic operation with currency values and currency literals	`If an account's annual revenue divided by 1000000 in "USD" is more than 1 then …`	Account BO Account BC Annual Revenue
function	currency-value-divided-by-number	Arithmetic operation with currency values and currency literals	`If an account's total potential volume divided by 12 is more than the account's revenue then …`	Account BO Account BC Total Potential Volume
function	currency-value-divided-by-value	Arithmetic operation with currency values and currency literals	`Set "potential/current volume" of an account to the account's total potential volume divided by current volume`	Account BO Account BC Current Volume Total Potential Volume
function	currency-value-minus-literal	Arithmetic operation with currency values and currency literals	`If an account's current volume minus 100000 in "USD" is at least the account's total potential volume then …`	Account BO Account BC Current Volume Total Potential Volume

Table 18. Siebel Actions, Functions, and Predicates in HaleyAuthority

Action type	Action	Purpose	Example of Phrasing	Example Requires Importing These Objects:
function	currency-value-minus-value	Arithmetic operation with currency values and currency literals	Add an account's total potential volume minus the account's current volume for the account	Account BO Account BC Current Volume Total Potential Volume
function	currency-value-plus-literal	Arithmetic operation with currency values and currency literals	Set "Credit Auto Approval Limit" of an account to the account's credit auto approval limit plus 500000 in "USD"	Account BO Account BC Credit Auto Approval Limit
function	currency-value-plus-value	Arithmetic operation with currency values and currency literals	Add an account's credit used plus the account's credit available amount for the account	Account BO Account BC Credit Used Credit Auto Approval Limit
function	currency-value-times-number	Arithmetic operation with currency values and currency literals	Set "Annual Revenue" of an account to the account's revenue times 12	Account BO Account BC Revenue
function	get-active-view-name	Get the name of the current active view	If active view name is "Account List View" then …	Does not depend on any object import.
function	get-profile-attribute	Get a profile attribute	If profile attribute of "Hobby" is "Fishing" then …	Does not depend on any object import.

Table 18. Siebel Actions, Functions, and Predicates in HaleyAuthority

Action type	Action	Purpose	Example of Phrasing	Example Requires Importing These Objects:
predicate	currency-at-least	Comparison predicate involving currency values and currency literals	`If an account's current volume is at least the account's total potential volume then …`	Account BO Account BC Current Volume Total Potential Volume
predicate	currency-at-least-one-literal	Comparison predicate involving currency values and currency literals	`If an account's revenue is at least 3000000 in "USD" then …`	Account BO Account BC Revenue
predicate	currency-at-least-two-literals	Comparison predicate involving currency values and currency literals	`If 100 in "USD" is at least 100 in "EUR" then …`	Does not depend on any object import.
predicate	currency-at-most	Comparison predicate involving currency values and currency literals	`If an account's current volume is at most the account's total potential volume then …`	Account BO Account BC Current Volume Total Potential Volume
predicate	currency-at-most-one-literal	Comparison predicate involving currency values and currency literals	`If an account's revenue is at most 3000000 in "USD" then …`	Account BO Account BC Revenue
predicate	currency-at-most-two-literals	Comparison predicate involving currency values and currency literals	`If 100 in "USD" is at most 100 in "EUR" then …`	Does not depend on any object import.

Table 18. Siebel Actions, Functions, and Predicates in HaleyAuthority

Action type	Action	Purpose	Example of Phrasing	Example Requires Importing These Objects:
predicate	currency-equal-to	Comparison predicate involving currency values and currency literals	`If an account's current volume is equal to the account's total potential volume then …`	Account BO Account BC Current Volume Total Potential Volume
predicate	currency-equal-to-one-literal	Comparison predicate involving currency values and currency literals	`If an account's revenue is equal to 3000000 in "USD" then …`	Account BO Account BC Revenue
predicate	currency-equal-to-two-literals	Comparison predicate involving currency values and currency literals	`If 100 in "USD" is equal to 100 in "EUR" then …`	Does not depend on any object import.
predicate	currency-less-than	Comparison predicate involving currency values and currency literals	`If an account's current volume is less than the account's total potential volume then …`	Account BO Account BC Current Volume Total Potential Volume
predicate	currency-less-than-one-literal	Comparison predicate involving currency values and currency literals	`If an account's revenue is less than 3000000 in "USD" then …`	Account BO Account BC Revenue
predicate	currency-less-than-two-literals	Comparison predicate involving currency values and currency literals	`If 100 in "USD" is less than 100 in "EUR" then …`	Does not depend on any object import.

Table 18. Siebel Actions, Functions, and Predicates in HaleyAuthority

Action type	Action	Purpose	Example of Phrasing	Example Requires Importing These Objects:
predicate	currency-more-than	Comparison predicate involving currency values and currency literals	`If an account's current volume is more than the account's total potential volume then …`	Account BO Account BC Current Volume Total Potential Volume
predicate	currency-more-than-one-literal	Comparison predicate involving currency values and currency literals	`If an account's revenue is more than 3000000 in "USD" then …`	Account BO Account BC Revenue
predicate	currency-more-than-two-literals	Comparison predicate involving currency values and currency literals	`If 100 in "USD" is more than 100 in "EUR" then …`	Does not depend on any object import.
predicate	user-in-taskmode	Test whether the user is in task mode	`If user is in task mode then …`	Does not depend on any object import.

Rule Tables in the Siebel Runtime Database

Table 19 describes the tables in a Siebel runtime database which are used to store rules metadata that is used during runtime.

Table 19. Tables in the Runtime Database

Table	Purpose
S_BR_MODULE	This table stores rule module name and configuration data - rule module name, business object name, data assertion mode, inconsistent flag, status, and comment. A record is created when a new rule module is deployed from the Deployment plug-in. The configuration data is updated through the Administration - Business Rules screen in the Siebel application.
S_BR_GBL_PARM	This table stores information about the knowledge base - model dirty flag and ruleset name. This data is used internally only and is not visible to the user.
S_BR_GBL_BINARY	This table stores model files - model.bin, unauthoredRules.bin, unauthoredSiebelRules.bin, unauthoredDiagnostics.bin, and measures.xml. These columns contain the semantic model data (concepts, relations, some basic rules, and so on). The records are created in this table by the Deployment plug-in when you deploy rule modules and check the box *Update model when deploying selected modules* on the Siebel Deployment dialog. The data in this table is used internally and is not visible to the user.
S_BR_MODULE_BIN	This table stores compiled rule modules. The records are created in this table by the Deployment plug-in when you deploy a rule module. The data in this table is used internally and is not visible to the user.
S_BR_MOD_BC_REL	This table stores business component (including multi-value group) relations configured for each rule module. The records are created or deleted when a user creates or deletes rows in the Rule Module Relations applet on the Rule Module List View. The data in this table is used to query runtime data for rule execution.
S_BR_MDL_BC_REL	This table stores business component relations imported for each business object. The records are created by the object importer when a user imports a business component hierarchy for a business object. The records are visible to the user in the pick applet on the Business Component field of the Rule Module Relations applet in the Rule Module List View.

Table 19. Tables in the Runtime Database

Table	Purpose
S_BR_MDL_BC_FLD	This table stores imported fields for each imported business component. The records are created by the object importer when a user imports fields for a business component. The records are not visible to the user and are used internally at runtime when querying the runtime data for execution.
S_BR_CONCEPT	This table stores the HaleyAuthority concept id for each imported business component, field, and multi-value group. The records are created by the object importer, and used by the runtime, and not visible to the user.
S_BR_KNWLDGBASE	This table is intended to provide support for multiple knowledge bases in future releases. For Siebel and Haley releases that support only one knowledge base for each implementation, this table contains only one record for the single knowledge base. The record is not visible to the user.

Siebel Runtime Events Supported by Rules

The business component runtime events in which you can implement rules are:

- Associate
- ChangeRecord
- CopyRecord
- InvokeMethod
- NewRecord
- PreAssociate
- PreCopyRecord
- PreDeleteRecord
- PreInvokeMethod
- PreNewRecord
- PreQuery
- PreSetFieldValue
- PreWriteRecord
- Query
- SetFieldValue
- WriteRecord

- WriteRecordNew
- WriteRecordUpdated

Input Property Set Schema for Business Rule Service

The following XSD (XML Schema Definition) represents the complete structure of the input property set that is expected by the Business Rule Service business service. If you use the Siebel EAI XML Write to File service to export any Business Rule Service input property set to XML, then the structure of that XML file matches the structure described in the following XSD.

For more information about the EAI XML Write to File service, see *XML Reference: Siebel Enterprise Application Integration*.

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

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://
www.siebel.com/xml/BizRule" targetNamespace="http://www.siebel.com/xml/BizRule"
elementFormDefault="qualified" attributeFormDefault="unqualified">

   <xs:element name="PropertySet" type="BizRuleInputPropertySetType">

      <xs:annotation>

         <xs:documentation xml:lang="en">

         ************************************************************************

         This describes the format of Input PropertySet passed to the

         CSSBizRuleService.

         The following 3 properties can be passed.

            RuleModuleName - name of the rule module the caller wants to evaluate

            GetMoreData - whether to get more children business component data

                        using search specifications defined with the rule module

                        if necessary, or to evaluate the rule module with the

                        data passed in.

                        This is an optional property and the default is "N".

            PerformAction - whether to carry out the result, i.e., whether to

                        perform setfieldvalue actions.

                        This is an optional property and the default is "N".

         ************************************************************************
```

```
            </xs:documentation>

         </xs:annotation>

      </xs:element>

      <xs:complexType name="BizRuleInputPropertySetType">

         <xs:sequence minOccurs="0" maxOccurs="unbounded">

            <xs:element name="BusCompList" type="BusCompListType"/>

         </xs:sequence>

         <xs:attribute name="RuleModuleName" type="xs:string" use="required"/>

         <xs:attribute name="GetMoreData" type="BooleanType" use="optional"

         default="N"/>

         <xs:attribute name="PerformAction" type="BooleanType" use="optional"
   default="N"/>

      </xs:complexType>

      <xs:complexType name="BusCompListType">

         <xs:sequence maxOccurs="unbounded">

            <xs:element name="BusComp" type="BusCompType"/>

         </xs:sequence>

         </xs:complexType>

      <xs:complexType name="BusCompType">

         <xs:sequence>

            <xs:element name="Field" type="FieldType" minOccurs="0"

            maxOccurs="unbounded"/>

            <xs:element name="BusCompList" type="BusCompListType" minOccurs="0"

            maxOccurs="unbounded"/>

         </xs:sequence>

         <xs:attribute name="Name" type="xs:string" use="required"/>

         <xs:attribute name="Id" type="xs:string" use="required"/>

         <xs:attribute name="IsPrimary" type="BooleanType" use="optional"/>

         <xs:attribute name="MVLinkName" type="xs:string" use="optional"/>

      </xs:complexType>
```

```
    <xs:complexType name="FieldType" mixed="true">
       <xs:attribute name="Name" type="xs:string" use="required"/>
       <xs:attribute name="Type" type="FieldDataType" use="required"/>
       <xs:attribute name="CurrencyCode" type="xs:string" use="optional"/>
       <xs:attribute name="ExchangeRateDate" type="xs:string" use="optional"/>
    </xs:complexType>
    <xs:simpleType name="FieldDataType">
       <xs:restriction base="xs:string">
          <xs:enumeration value="DTYPE_BOOL"/>
          <xs:enumeration value="DTYPE_CURRENCY"/>
          <xs:enumeration value="DTYPE_ID"/>
          <xs:enumeration value="DTYPE_INTEGER"/>
          <xs:enumeration value="DTYPE_NOTE"/>
          <xs:enumeration value="DTYPE_NUMBER"/>
          <xs:enumeration value="DTYPE_PHONE"/>
          <xs:enumeration value="DTYPE_TEXT"/>
          <xs:enumeration value="DTYPE_UTCDATETIME"/>
          <xs:enumeration value="DTYPE_DATE"/>
          <xs:enumeration value="DTYPE_TIME"/>
       </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="BooleanType">
       <xs:restriction base="xs:string">
          <xs:enumeration value="Y"/>
          <xs:enumeration value="N"/>
       </xs:restriction>
    </xs:simpleType>
 </xs:schema>
```

Output Property Set Schema for Business Rule Service

The following XSD (XML Schema Definition) represents the complete structure of the output property set that is generated by the Business Rule Service business service. If you use the Siebel EAI XML Write to File service to export any Business Rule Service output property set to XML, then the structure of that XML file matches the structure described in the following XSD.

For more information about the EAI XML Write to File service, see *XML Reference: Siebel Enterprise Application Integration*.

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

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="http://
www.siebel.com/xml/BizRule" targetNamespace="http://www.siebel.com/xml/BizRule"
elementFormDefault="qualified" attributeFormDefault="unqualified">

   <xs:element name="PropertySet" type="BizRuleOnputPropertySetType">

      <xs:annotation>

         <xs:documentation xml:lang="en">

         ************************************************************************

         There are predefined set of results returned from the Rules Engine.

            Validation - The result of evaluating rules is either valid,

                         or invalid with reasons - validation messages.

            Derivation - The result of evaluating rules is one or more

                         values whether it's derived or not.

            The following predefined methods.

               SetFieldValue

               RaiseErrorText

               SetProfileAttribute

         ************************************************************************

         </xs:documentation>

      </xs:annotation>

   </xs:element>

   <xs:complexType name="BizRuleOnputPropertySetType">

      <xs:sequence>

         <xs:element name="ValidationList" type="ValidationListType"
```

```
            minOccurs="0"/>

            <xs:element name="DerivationList" type="DerivationListType"
            minOccurs="0"/>

            <xs:element name="SetFieldValueList" type="SetFieldValueListType"
            minOccurs="0"/>

            <xs:element name="RaiseErrorText" type="xs:string" minOccurs="0"/>

            <xs:element name="SetProfileAttributeList"
            type="SetProfileAttributeListType" minOccurs="0"/>

         </xs:sequence>

      </xs:complexType>

      <xs:complexType name="ValidationListType">

         <xs:sequence>

            <xs:element name="Validation" type="ValidationType"
            maxOccurs="unbounded"/>

         </xs:sequence>

      </xs:complexType>

      <xs:complexType name="DerivationListType">

         <xs:sequence>

            <xs:element name="Derivation" type="DerivationType"
            maxOccurs="unbounded"/>

         </xs:sequence>

      </xs:complexType>

      <xs:complexType name="SetFieldValueListType">

         <xs:sequence>

            <xs:element name="SetFieldValue" type="SetFieldValueType"
            maxOccurs="unbounded"/>

         </xs:sequence>

      </xs:complexType>

      <xs:complexType name="SetProfileAttributeListType">

         <xs:sequence>

            <xs:element name="SetProfileAttribute" type="SetProfileAttributeType"
            maxOccurs="unbounded"/>

         </xs:sequence>
```

```
</xs:complexType>

<xs:complexType name="ValidationType">

   <xs:sequence maxOccurs="unbounded">

      <xs:element name="Message" type="xs:string" minOccurs="0"
      maxOccurs="unbounded"/>

   </xs:sequence>

   <xs:attribute name="Result" type="ValidationResultType" use="required"/>

   <xs:attribute name="BusCompName" type="xs:string" use="required"/>

   <xs:attribute name="BusCompId" type="xs:string" use="required"/>

</xs:complexType>

<xs:simpleType name="ValidationResultType">

   <xs:restriction base="xs:string">

      <xs:enumeration value="Valid"/>

      <xs:enumeration value="Invalid"/>

      <xs:enumeration value="Inconsistent"/>

   </xs:restriction>

</xs:simpleType>

<xs:complexType name="DerivationType">

   <xs:sequence>

      <xs:element name="Value" type="DerivationValueType"
      maxOccurs="unbounded"/>

   </xs:sequence>

   <xs:attribute name="BusCompName" type="xs:string" use="required"/>

   <xs:attribute name="BusCompId" type="xs:string" use="required"/>

</xs:complexType>

<xs:complexType name="DerivationValueType" mixed="true">

   <xs:attribute name="CurrencyCode" use="optional"/>

   <xs:attribute name="ExchangeRateDate" use="optional"/>

</xs:complexType>

<xs:complexType name="SetFieldValueType">

   <xs:sequence>
```

```
            <xs:element name="Field" type="FieldType" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="BusCompName" type="xs:string" use="required"/>
            <xs:attribute name="BusCompId" type="xs:string" use="required"/>
        </xs:complexType>
        <xs:complexType name="FieldType" mixed="true">
            <xs:attribute name="Name" type="xs:string" use="required"/>
            <xs:attribute name="CurrencyCode" type="xs:string" use="optional"/>
            <xs:attribute name="ExchangeRateDate" type="xs:string" use="optional"/>
        </xs:complexType>
        <xs:complexType name="SetProfileAttributeType">
            <xs:attribute name="Name" type="xs:string" use="required"/>
            <xs:attribute name="Value" type="xs:string" use="required"/>
        </xs:complexType>
    </xs:schema>
```

The following example is an output property set consisting of ValidationList only:

```
    <?xml version="1.0" encoding="UTF-8" ?>
    <?Siebel-Property-Set EscapeNames="true"?>
    <PropertySet>
        <ValidationList>
            <Validation BusCompId="6-4AH4A" Result="Valid" BusCompName="Expense" />
        </ValidationList>
        </PropertySet>
```

The following example is an output property set consisting of DerivationList and SetFieldValueList:

```
    <?xml version="1.0" encoding="UTF-8" ?>
    <?Siebel-Property-Set EscapeNames="true"?>
    <PropertySet>
        <DerivationList>
            <Derivation BusCompId="6-4AEEW" BusCompName="Opportunity">
                <Value>Excellent</Value>
```

```
      </Derivation>
    </DerivationList>
    <SetFieldValueList>
      <SetFieldValue BusCompId="6-4AEEW" BusCompName="Opportunity">
        <Field Name="Quality">1-Excellent</Field>
      </SetFieldValue>
    </SetFieldValueList>
  </PropertySet>
```

The following example is an output property set consisting of SetProfileList only:

```
  <?xml version="1.0" encoding="UTF-8" ?>
  <?Siebel-Property-Set EscapeNames="true"?>
  <PropertySet>
    <SetProfileAttributeList>
      <SetProfileAttribute Value="eft" Name="abc" />
    </SetProfileAttributeList>
  </PropertySet>
```

B Troubleshooting Siebel Business Rules

This appendix provides technical support guidelines and troubleshooting information. The appendix contains the following topics:

About Technical Support for Rules

You receive Haley authoring and runtime software as part of Oracle's Siebel Tools installation. Besides creating and deploying HaleyAuthority rules for Siebel objects, you can also use HaleyAuthority to create and deploy rules for other applications.

Oracle Corporation provides technical support for HaleyAuthority rules that are intended for use with Siebel objects and applications as described in this document. Oracle Corporation provides technical support only for scenarios that observe the following guidelines on the HaleyAuthority knowledge base in which rule statements for Siebel objects are written:

- The HaleyAuthority knowledge base does not reside in the Siebel runtime database or in the Siebel Master Repository database.
- The HaleyAuthority knowledge base contains no custom concepts other than concepts that Siebel Object Importer creates.
- The HaleyAuthority knowledge base contains no custom relations or phrasings among concepts other than relations and phrasings that Siebel Object Importer creates.
- No concepts, relations, or phrasings that Siebel Object Importer creates are deleted from the HaleyAuthority knowledge base.

As with other aspects of Siebel applications, implementation assistance is available through Professional Services.

Troubleshooting Rules Configuration and Activation

This topic provides information for resolving issues that occur when you configure and activate a rule module in the Rule Modules List view of your Siebel application's Administration - Business Rules screen.

To resolve a problem, look for the problem in the list of Symptoms/Error messages in Table 20.

Table 20. Troubleshooting Rules Configuration and Activation

Symptoms/ Error Messages	Diagnostic Steps/Cause	Solution
Inconsistent flag has value Y.	This read-only flag is set in the Rule Modules applet when the rule module is inconsistent with the currently deployed semantic role model. This state occurs when you deploy the rule module from HaleyAuthority, then the semantic role model is changed and redeployed, which may cause your rule module to be inconsistent with the model. Your rule then contains statements that are not understood by HaleyAuthority. One sample scenario for which this state occurs is: ■ The module uses a field as a string in rule statements, then somebody changes the field from DTYPE_TEXT to DTYPE_BOOL in the in the Siebel Repository. The field is re-imported or the knowledge base is synchronized with the Siebel Repository. The model is re-deployed with another rule module. Your rule module is then inconsistent with the currently deployed semantic role model. Rule statements using the field are inconsistent with the currently understood statements that use that field entity.	1 In your Siebel application, navigate to Administration - Business Rules > Rule Modules List. 2 Deactivate the inconsistent rule module. For information on deactivating a rule module, see "Using the Deactivate Module Option" on page 65 3 In HaleyAuthority, edit rule statements to make them understood. 4 Re-deploy the rule module. 5 In your Siebel application's Rule Modules List view, confirm that the rule module's Inconsistent flag has the value N, then re-configure and activate the rule module.

Troubleshooting Rules Implementation

This topic provides information for resolving issues that are particular to implementing rule modules in various contexts - runtime events, Siebel workflows, Siebel Task UI flows, or script.

Table 21. Troubleshooting Rules Implementation

Context	Symptoms/Error Messages	Diagnostic Steps/Cause	Solution
Runtime event	At runtime, you get an error message that the deployed rule does not exist.	The name of the rule module in the Business Service Context field for the runtime event's Action must be identical to the name of the rule module in HaleyAuthority. The comparison is case-sensitive and space-sensitive.	Correct the rule module name in the Business Service Context field for the runtime event's Action. For more information, see "Repeat Step 4 through Step 5 for the Service Request Activity - PreWriteRecord module. In the Rule Module Relation applet, create a new record as follows:" on page 88.
Runtime event	At runtime, you get an error message that the deployed rule does not exist.	The name of the rule module should not include any special characters, such as *, ?, /, !, @ or quotes.	1 Deactivate the rule module. 2 In HaleyAuthority, rename the rule module. 3 Redeploy the module. 4 Activate the module, which now appears as a new module. 5 Enter the corrected module name in the Business Service Context field for the runtime event's Action. For information on each of these steps, see "Scenario for Using Rules to Validate Data at Runtime" on page 78.

Troubleshooting Rules Output

This topic provides information for debugging problems with the Business Rule Service output property set.

TIP: You can use the WritePropSet method of the Siebel EAI XML Write to File service to print out the output property set returned from Business Rule Service. Use an XML viewier to examine the output.

For more information about the Oracle's Siebel EAI XML Write to File service, see *XML Reference: Siebel Enterprise Application Integration*.

Index