

# CA Aion<sup>®</sup> Business Rules Expert

## Developer Overview

r11



This documentation and any related computer software help programs (hereinafter referred to as the "Documentation") are for your informational purposes only and are subject to change or withdrawal by CA at any time.

This Documentation may not be copied, transferred, reproduced, disclosed, modified or duplicated, in whole or in part, without the prior written consent of CA. This Documentation is confidential and proprietary information of CA and may not be used or disclosed by you except as may be permitted in a separate confidentiality agreement between you and CA.

Notwithstanding the foregoing, if you are a licensed user of the software product(s) addressed in the Documentation, you may print a reasonable number of copies of the Documentation for internal use by you and your employees in connection with that software, provided that all CA copyright notices and legends are affixed to each reproduced copy.

The right to print copies of the Documentation is limited to the period during which the applicable license for such software remains in full force and effect. Should the license terminate for any reason, it is your responsibility to certify in writing to CA that all copies and partial copies of the Documentation have been returned to CA or destroyed.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CA PROVIDES THIS DOCUMENTATION "AS IS" WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT. IN NO EVENT WILL CA BE LIABLE TO THE END USER OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USE OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, LOST INVESTMENT, BUSINESS INTERRUPTION, GOODWILL, OR LOST DATA, EVEN IF CA IS EXPRESSLY ADVISED IN ADVANCE OF THE POSSIBILITY OF SUCH LOSS OR DAMAGE.

The use of any software product referenced in the Documentation is governed by the applicable license agreement and is not modified in any way by the terms of this notice.

The manufacturer of this Documentation is CA.

Provided with "Restricted Rights." Use, duplication or disclosure by the United States Government is subject to the restrictions set forth in FAR Sections 12.212, 52.227-14, and 52.227-19(c)(1) - (2) and DFARS Section 252.227-7014(b)(3), as applicable, or their successors.

Copyright © 2010 CA. All rights reserved. All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.

## CA Product References

This document references the following CA products:

- CA Aion® Business Rules Expert (CA Aion BRE)

## Contact CA

### Contact Technical Support

For your convenience, CA provides one site where you can access the information you need for your Home Office, Small Business, and Enterprise CA products. At <http://ca.com/support>, you can access the following:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

### Provide Feedback

If you have comments or questions about CA product documentation, you can send a message to [techpubs@ca.com](mailto:techpubs@ca.com).

If you would like to provide feedback about CA product documentation, complete our short [customer survey](#), which is also available on the CA Support website, found at <http://ca.com/docs>.



# Contents

---

<b>Chapter 1: Introduction to CA Aion BRE</b>	<b>9</b>
CA Aion BRE Architecture .....	9
Business Client User and Developer Roles .....	10
 <b>Chapter 2: Basic Rule Programming Techniques</b>	 <b>13</b>
Create and Maintain Static Rules .....	13
Rules and Rule Methods .....	14
Open the Method Editor .....	14
Open the Rule Editor .....	15
The Decision Table Editor .....	18
Use the Decision Table Editor .....	18
Benefits of Decision Tables .....	21
Post Rules .....	21
View an Inference Block .....	22
Call Inference Methods .....	23
The Rule Analyzer .....	24
Analyze Rule Processing .....	25
 <b>Chapter 3: Access to Essential Databases</b>	 <b>29</b>
The DataLib System Library .....	29
Specify a Data Source .....	29
Test the Data Connection .....	31
Create a Query Class .....	32
Execute the Discount Application in the Aion IDE .....	34
Execute the Discount Application in the Aion IDE .....	37
 <b>Chapter 4: Develop an Aion BRE Knowledge Base</b>	 <b>39</b>
The Purchase Checker System .....	39
System Specification .....	39
Requirements .....	39
Design Assumptions .....	40
Build the Purchase Checker System .....	40
Create the Application .....	41
Connect to the Database .....	42
Create a Query Class .....	43
Map the Query Class to the Database .....	45

---

Test the Database Connection .....	45
View the Database in the Aion Debugger .....	47
Create Attributes to Store Credit Limit and Approval .....	49
Create a Method to Calculate Credit Limit .....	50
Create a Decision Table .....	52
Post the Decision Table .....	57
Call the Decision Table .....	58
Test the Decision Table .....	60
Rules to Approve Purchases .....	62
Write the Approval Rule .....	62
Write the Disapproval Rule .....	64
Post the Rules .....	65
Call and Test the Rules .....	66
Report the Approval/Disapproval Decision .....	67
Generate Message Text .....	67
Display a Message Box .....	68
Remove an Unused Library (Optional) .....	69

## **Chapter 5: Build and Deploy Applications and Components** **71**

Build a CA Aion Application under Windows .....	71
Specify Component Interfaces .....	72
Choose the Type of Component to Build .....	73
Build the Component .....	73
Develop Applications for UNIX Deployment .....	74
Develop Applications Overview .....	74
Build Applications with AionBuilder .....	75
Build Applications with Command Line Utilities .....	77
The Respawn Utility .....	77
The Reexec Utility .....	78
Compilation Hints and Suggestions .....	79

## **Appendix A: The Aion BRE Inference Engine and Rules** **81**

Basic Inferencing Techniques .....	81
Rule Structure .....	82
Example .....	82
Forward Chaining .....	82
Example .....	83
Backward Chaining .....	83
Example .....	84
Aion BRE Rule Types and Syntax .....	84

---

<b>Appendix B: The Aion BRE Debugger</b>	<b>87</b>
The Debugger Button .....	87
The Debugger Window .....	87
Open the Debugger Window .....	89
The Debugger Toolbar .....	89
Step through Methods.....	89
Set Breakpoints and Watchpoints .....	90
 <b>Index</b>	 <b>91</b>





# Chapter 1: Introduction to CA Aion BRE

---

CA Aion Business Rules Expert (CA Aion BRE) can help you develop complex business applications that drive critical functions, and adapt them for a changing marketplace.

## CA Aion BRE Architecture

CA Aion BRE is an application construction tool that brings business intelligence to business clients. CA Aion BRE provides an integrated development environment, the CA Aion Integrated Development Environment (Aion IDE), which addresses the needs of the IT developer and allows the business clients to maintain the business rules of an Aion application. For more information, see the section Business Client User and Developer Roles.

At the core of CA Aion BRE is support for the principles of object-oriented programming. This product supports common object-oriented concepts such as classes, inheritance, and polymorphism. CA Aion BRE extends traditional object orientation with sophisticated mechanisms that are not found in most commercially available object-oriented programming languages.

Behind the CA Aion BRE object orientation lie database connectivity and the product's powerful Aion BRE inference engine. CA Aion BRE provides native connectivity to most popular database management and to a number of common PC database management systems through ODBC. The CA Aion BRE inference engine supports a broad spectrum of business rule types to meet business needs for inferencing, including decision tables, simple IF/THEN rules, powerful rules that operate over sets of business objects, and trigger rules (called *daemon rules*). Rules for an Aion application may be either compiled into the application or, in the case of decision table rules, loaded dynamically at runtime.

A CA Aion BRE application can be related to the outside world in many ways. CA Aion BRE's own graphical user interface (GUI) tools can be used to create a user interface to the application. The application can also be deployed as a standalone application without a user front-end. That is, it can be deployed as a "batch" program. In addition, CA Aion BRE provides mechanisms to create component interfaces to common languages, interfacing standards, and messaging services on the market today, including Java, COM/MTS, C/C++, web services, IBM's Websphere MQ Series, and others.

With CA Aion BRE, you can request that your application be developed with a specific component interface, and CA Aion BRE will do all the work to generate that interface layer. CA Aion BRE applications can be deployed under a wide variety of operating systems, including Windows, UNIX, Linux, OS/390, and z/OS.

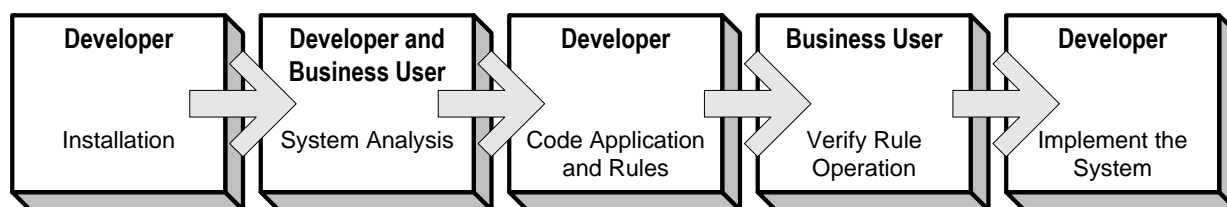
**More Information:**

Rules and inferencing, see the *CA Aion BRE Rules Guide*.

CA Aion BRE Objects, see the "Overview of CA Aion BRE Objects" in the *CA Aion BRE Product Guide*

## Business Client User and Developer Roles

Developing a business intelligence application with CA Aion BRE is a joint effort between the business client and the (IT) developer. CA Aion BRE supports several models to structure this joint effort. The classic model is depicted in the following diagram.



According to this model, the developer and business user meet to discuss the business problem and to decide on the rules that solve the problem and how these rules should function in the system. From this discussion, the developer codes the application and rules. The business user verifies the operation of the system to ensure that the rules function as planned. Once rules are approved, the developer implements the system.

This classic model has two well-known drawbacks:

- It requires the developer to become an expert in the business problem.
- Business rule changes require the IT developer to recode and recompile.

CA Aion BRE also lets business clients create and maintain (manage) the business rules without requiring the developer to become a business rules expert. Under these models, the IT developer codes only a skeletal Aion application that defines the business objects and performs procedural chores. Coding the business rules shifts to the business client.

The Dynamic Rule Manager allows business clients to create and maintain business rules using a purely business-oriented vocabulary. The business rules reside in a rule base that exists outside of the CA Aion application. Because dynamic rules do not require recompilation of the Aion application, dynamic rules are especially useful for applications that run continuously and require frequent rule changes.



# Chapter 2: Basic Rule Programming Techniques

---

Static rules allow programmers to express the business knowledge needed to complete a task, because this knowledge is coded in declarative statements. This chapter describes how to build and maintain these rules using various methods, and looks at using static rules to take advantage of the Aion BRE inference engine. The Rule Editor and the Decision Table Editor are introduced to assist in this task.

## Create and Maintain Static Rules

In traditional procedural programming, application developers must specify the exact flow from one statement to the next. Developers use complex looping instructions, conditional statements, and algorithms.

CA Aion BRE lets developers separate control logic from business knowledge, which is coded into a declarative rules set. Because the rules are simpler and easier to maintain, CA Aion BRE allows for productivity gains over traditional procedural programming.

The Aion BRE inference engine processes rules that are relevant to solving the immediate business problem. When you execute an Aion application, the inference engine follows an algorithm set that automatically decides which rules to use and when.

You can create two types of rules: static and dynamic. *Static rules* are hard-coded in an Aion application, and *dynamic rules* are often stored in a database (rulebase) and loaded when an application runs.

**Note:** This chapter provides details about static rules only. For more information on dynamic rules, see the *CA Aion BRE Rules Guide*.

## Rules and Rule Methods

CA Aion BRE provides several ways to represent rules, including basic IF-THEN statements, called *IFRULEs*, and decision tables as well as pattern matching rules, called *IFMATCH* rules, and *demons* (WHEN and WHENMATCH rules). The Aion BRE inference engine processes these rules using either forward or backward chaining. In the Discount application, for example, *IFRULEs* and decision tables are executed to determine pricing discounts for an order. Based on submitted information, the Aion BRE inference engine follows a chain of rules until it reaches a conclusion.

**Note:** For information about the types of rules supported in CA Aion BRE and the forward and backward chaining algorithms on the Aion BRE inference engine, see Basic Inferencing Techniques in the appendix “The Aion BRE Inference Engine and Rules.”

When you want to create a new static or dynamic rule, use an Assistant to step you through the necessary screens. Choose Tools, Assistants, Rule Assistant *or* Dynamic Rule Assistant.

Methods that include rules are called *rule methods*, and can be used in the same way as procedure-based methods. You can call rule methods from other objects, and inherit and specialize rule methods. This allows CA Aion BRE to handle large, complex applications. Developers can utilize the classification and modularization capabilities that objects provide and extend that with the ability of rules to capture complex, dynamic, and knowledge-intensive logic.

CA Aion BRE provides two editors for creating and modifying *IFRULEs*: the Rule Editor and the Method Editor. You have already been introduced to the Method Editor. Now you will see how to use it to maintain rules. This section also introduces the special Rule Editor that provides a structured environment for creating and maintaining rules.

### Open the Method Editor

Open BRE IDE from start menu and open discount.app from examples\Discount folder.

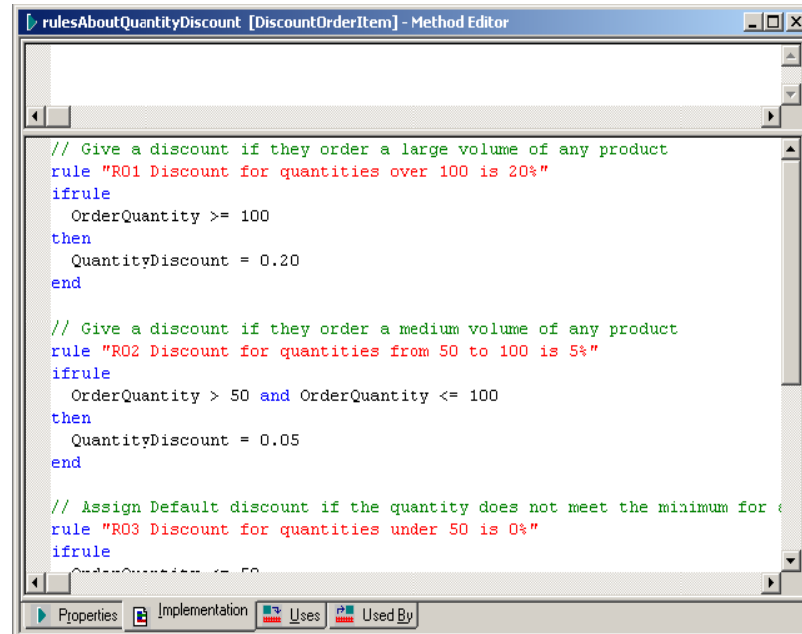
The Method Editor uses English-like syntax for the rules and allows developers to enter the rule text directly.

#### To open the Method Editor

1. In the Project Workplace, go to the Rules page and expand the DiscountOrderItem class.

- Double-click the rule method named RulesAboutQuantityDiscount.

The Method Editor appears showing the rules contained in the RulesAboutQuantityDiscount method.



- Close the Method Editor.

If you are prompted to save the changes, click No.

**Note:** For more information about the syntax used to define rules in the Method Editor, see Basic Inferencing Techniques in the appendix "The Aion BRE Inference Engine and Rules."

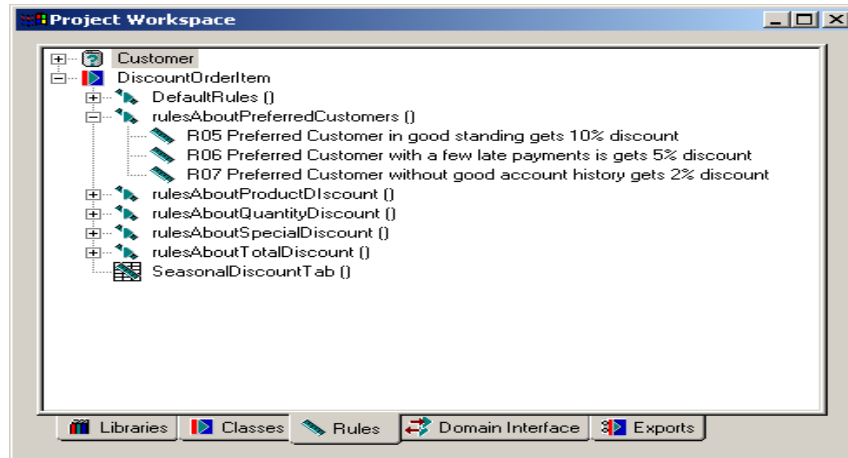
## Open the Rule Editor

The Rule Editor provides a structured environment for creating and editing rules. The Rule Editor presents the appropriate text boxes corresponding to the structural parts of the different types of rules that CA Aion BRE provides. In the Rule Editor, you can also browse the rules in the entire rule method as well as set the properties of the rule method.

**Note:** When you want to create a new static or dynamic rule, use an Assistant to step you through the necessary screens. Choose Tools, Assistants, Rule Assistant, or Dynamic Rule Assistant.

### To use the Rule Editor

1. In the Project Workspace, click the Rules tab.
2. Expand DiscountOrderItem to show the rule methods of this class.
3. Expand the RulesAboutPreferredCustomer rule method to show the individual rules in the method.



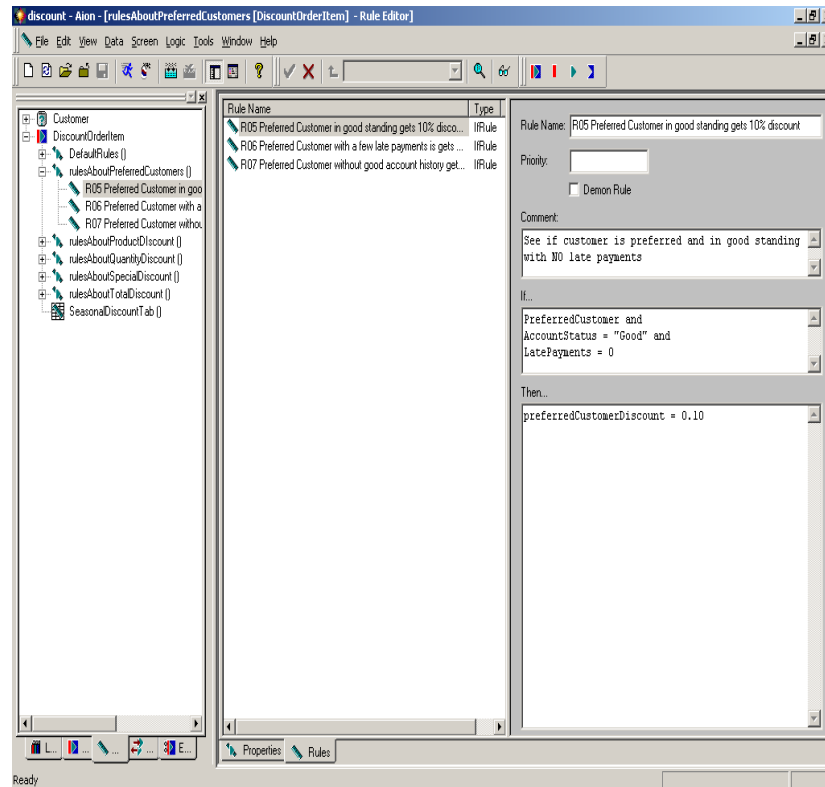


4. Right-click the R05 Preferred Customer in good standing gets a 10% discount rule.

A pop-up menu appears.

5. Choose Open.

The Rule Editor appears showing the selected rule.



6. Close the Rule Editor.
7. If you are prompted to save changes, click No.

**Note:** This rule is an IFRULE. A different structure is presented when defining or displaying an IFMATCH rule.

## The Decision Table Editor

Besides allowing you to code rules at the individual rule level, CA Aion BRE also allows you to express knowledge in the form of decision tables. *Decision tables* let you visualize a complete set of rules at one time. The inherent structure of a decision table prevents certain types of logical errors from going undetected, such as inconsistencies between rules within a rule set and incomplete reasoning paths.


**Note:** When you want to create a new decision table, use an Assistant to step you through the necessary screens. For static decision tables choose Tools, Assistants, Rule Assistant; for dynamic decision tables choose Tools, Assistants, Dynamic Rule Assistant.

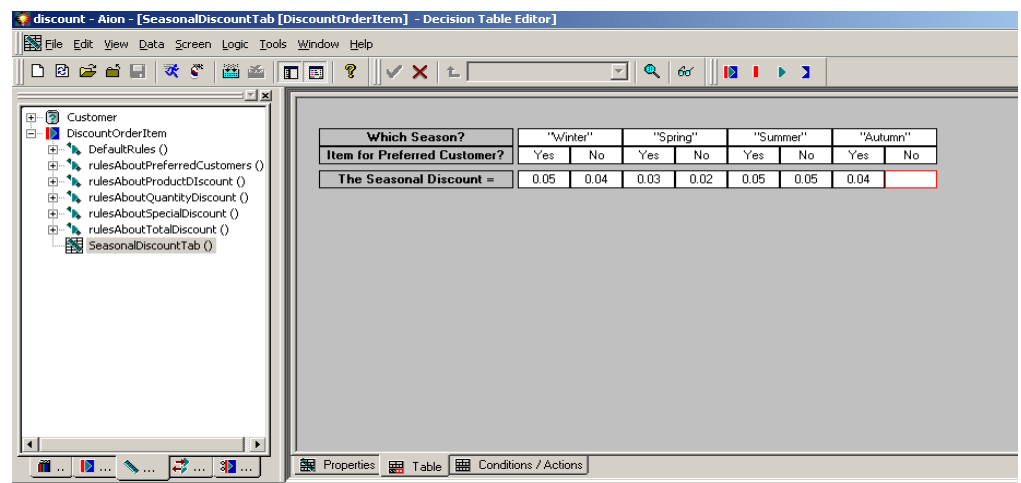
### Use the Decision Table Editor

The Decision Table Editor provides a visual structure for creating and modifying decision tables.

#### To open the Decision Table Editor

1. Expand the DiscountOrderItem on the Rules page as you did when accessing the Rule Editor.
2. Double-click the SeasonalDiscountTab rule method.

**Note:** You can easily identify a decision table by the decision table rule icon .

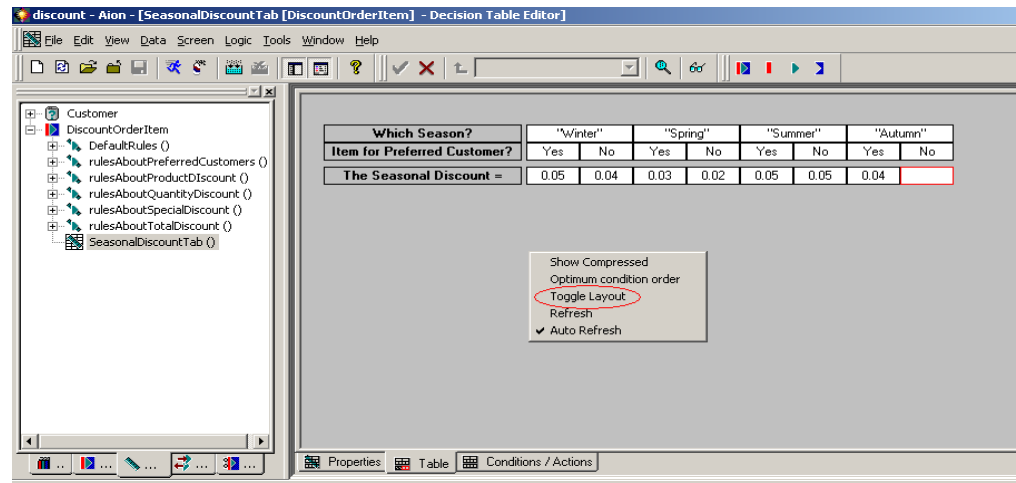


The decision table appears in the Decision Table Editor.

The default display of a decision table is called the landscape view. You will now change to portrait view.

3. In the Decision Editor, right-click in an empty space.

A pop-up menu appears:



4. Choose Toggle Layout.

The decision table appears in portrait view.

The screenshot shows the 'Decision Table Editor' window with the decision table in portrait orientation. The table structure is as follows:

Which Season?	Item for Preferred Customer?	The Seasonal Discount =
"Winter"	Yes	0.05
	No	0.04
"Spring"	Yes	0.03
	No	0.02
"Summer"	Yes	0.05
	No	0.05
"Autumn"	Yes	0.04
	No	

The top left column headings of the table shows the conditions (IF) and the cells beneath the conditions show all possible values for those conditions. In this example, the conditions are Which Season? and Item for Preferred Customer? The top right column heading of the table shows the actions (THEN). This table has one action, The Seasonal Discount =.

Study the SeasonalDiscountTab decision table and notice the following:

- Each action will execute based on a combination of conditions.
- When no actions are specified for a certain combination of conditions, a highlighted cell represents the possible logic error graphically.

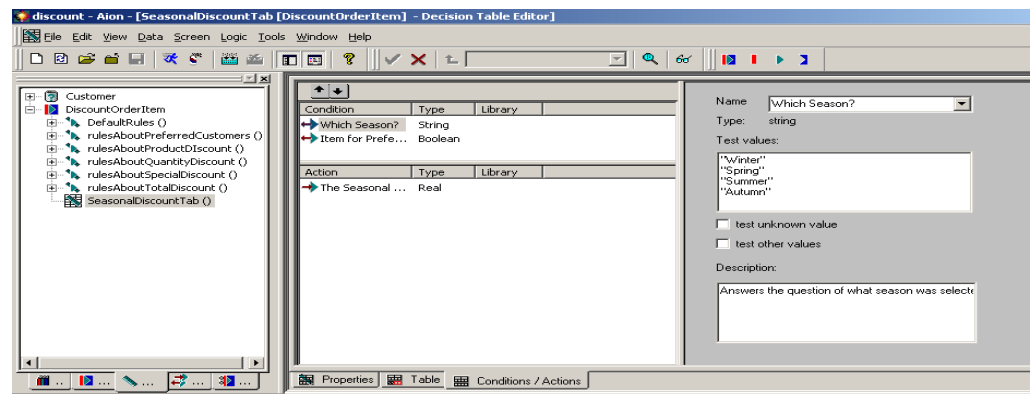
**Note:** To change information for actions, click the cell in the corresponding action column and enter the appropriate value.

- The first condition is the domain interface member, Which Season?. Remember, *domain interface members* are labels that represent methods. The inference engine uses the Which Season? label to reference and execute the Season() method. This method returns the season, which the inference engine compares to the values in the decision table. When the inference engine finds a value in the table that matches the value returned for Which Season?, the inference engine knows where in the table to continue processing.
- The action in this decision table, The Seasonal Discount =, is also a domain interface member. Find it on the Domain Interface tab and double-click it. The Method Editor appears, and its Properties page shows that the method SetSeasonalDiscount() is associated with The Seasonal Discount=. The input argument for SetSpecialDiscount() is the discount for the item. That input argument is specified in the cells beneath the action in the decision table.

5. Click on the Conditions/Actions tab.

You create and add or delete conditions and actions for a decision table on the Conditions/Actions page. You will use this page later in the section Build an Application, when you create your own decision table.

6. Click on one of the Conditions or Actions shown in the left-hand panels (in the following screen, we have clicked the Which Season? Condition).



7. Close the Decision Table Editor.

### More Information:

CA Aion BRE Product Guide.

## Benefits of Decision Tables

Using a decision table instead of a set of rules can offer the following benefits:

- Decision tables help ensure completeness and consistency, and make redundancy easier to avoid.
- Ensuring completeness means that all possible combinations of values are covered. The Decision Table Editor automatically displays all the possible combinations and shows any combinations for which no action is defined.
- Ensuring consistency means that you do not inadvertently define a condition more than once with different or incompatible actions. The inherent structure of a decision table prevents these kinds of errors from being committed.
- Avoiding redundancy means that you do not inadvertently define a condition more than once with the same action. The inherent structure of a decision table prevents these kinds of errors from being committed.
- CA Aion BRE decision tables include a runtime algorithm that ignores conditions that do not contribute to determining a specific action.
- Many business experts prefer to see decision processes represented as decision tables rather than as rules. Decision tables provide a simple view of the full picture, which may not be as easy to understand from a list of discrete rules.

## Post Rules

To use the rules in an Aion application, you must post those rules. Posting rules makes them available to the inference engine, which fires the rules according to the data that is being processed in the application.

Posting rules in CA Aion BRE focuses on rule methods. Recall that a rule method contains a list of rules or a single decision table. It consists of declarative code (rules) rather than procedural code (operations). A rule method is invoked the same way as a procedural method. The principal difference is that when a rule method executes, it only posts rules to the inference engine rather than executing code that effects some change in the state of the application.

A second difference between rule methods and procedural methods is that a rule method can be invoked only from within an inference block. *Inference blocks* are special structures within procedural methods that begin with INFER and end with END. These commands define the scope of an inference engine. Between INFER and END is a list of one or more rules and a ForwardChain or BackwardChain command.

**Note:** For more information about rule methods and posting rules, as well as inference blocks and invoking inferencing, see the *CA Aion BRE Rules Guide*.

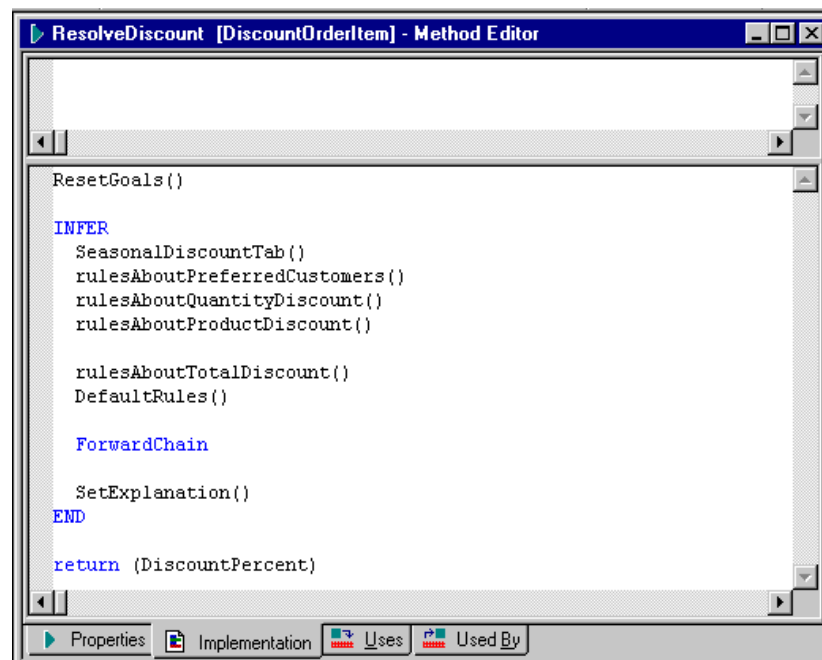
Inference blocks can also contain procedural code. Procedural methods that contain an inference block are called *inference methods*. Inference methods can contain multiple inference blocks as well as other code. An Aion application usually has several inference methods so that each method can handle a specific processing task and invoke only those rules that are appropriate to that task.

## View an Inference Block

### To view an inference block in the Discount application

1. On the Libraries page of the Project Workspace, expand the DiscountOrderItem class.
2. Double-click the ResolveDiscount() method.

The Method Editor displays the following code:



In the previous graphic, the INFER...END block of code; this defines this method as an inference method. The inference block calls six rule methods, each containing several rules related to a certain aspect of the Discount application. The ForwardChain command tells the inference engine to process the rules that have been posted to it in a forward chaining manner.

**Note:** For more information on forward chaining, see Basic Inferencing Techniques in the appendix "The Aion BRE Inference Engine and Rules."

Besides the rule methods, the inference block calls the `SetExplanation()` method. This call needs to be inside the inference block because the explanation displayed to users of the Discount application is based on the rule processing, which is retained only with the scope of the inference block.

**Note:** If you are not sure which methods in an application are inference methods, you can locate them by using the Aion Find feature. From the Edit menu, choose Find, and in the dialog that appears, search for the string "INFER" and select "Method" as the type of object to search. For more information about Find, click the Help button on the Find dialog.

## Call Inference Methods

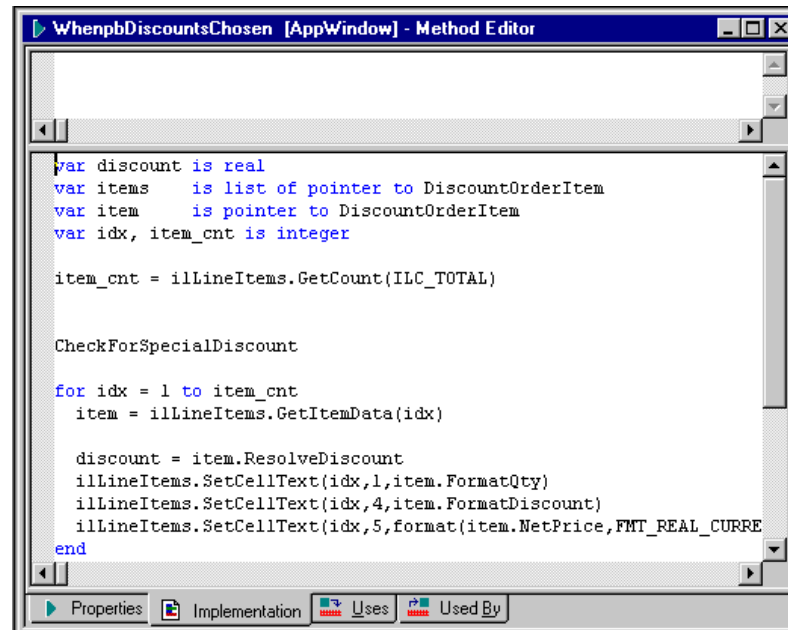
In Discount, three methods contain inference blocks:

- `ResolveDiscount()`
- `DetermineAccountStatus()`
- `CheckForSpecialDiscount()`

The users of Discount application invoke the inference method, `DetermineAccountStatus()`, when customer is selected from the customer drop down list. The users invoke the inference methods `CheckForSpecialDiscount()` and `ResolveDiscount()` when they click Resolve Discounts.

**To display a method that calls two of the inference methods in Discount**

1. In the Project Workspace, select the Library tab.
2. Expand the AppWindow class then double-click the WhenpbDiscountsChosen() method.



WhenpbDiscountsChosen() calls the CheckForSpecialDiscount() inference method first, and then calls ResolveDiscount() in a loop for each item entered into the application. This simple control strategy for calling inference methods checks for a special discount and then applies the discount.

3. Close all the editors.  
If you see the prompt to save changes, click No.

## The Rule Analyzer

CA Aion BRE provides a Rule Analyzer to help debug the chain of logic created in rule-based processing. The Rule Analyzer does this by showing the relationships between rules and attributes in a tree list format. You can choose to analyze in a forward or backward mode.

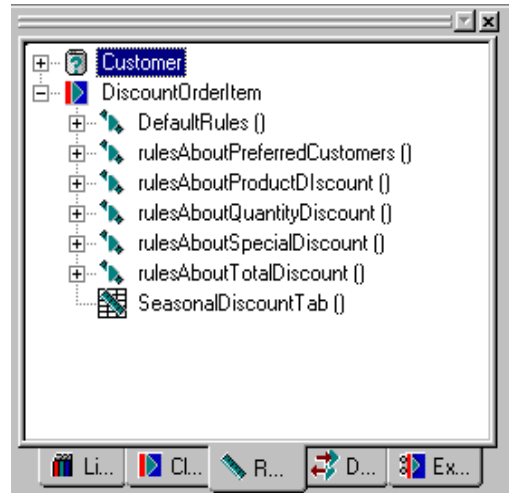
Use the forward mode to see which rules will fire when you assign a value to an attribute, and what attributes will receive values as a result of the rules firing. Use the backward mode to determine which rules can assign values to an attribute, and which attributes are needed for a rule to fire.



## Analyze Rule Processing

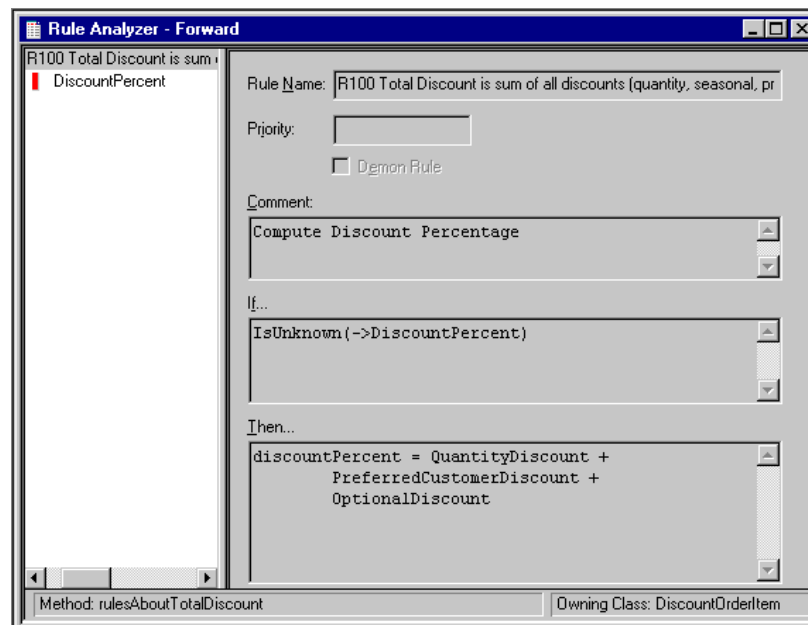
### To open the Rule Analyzer

1. On the Rules page of the Project Workspace, expand the class DiscountOrderItem.



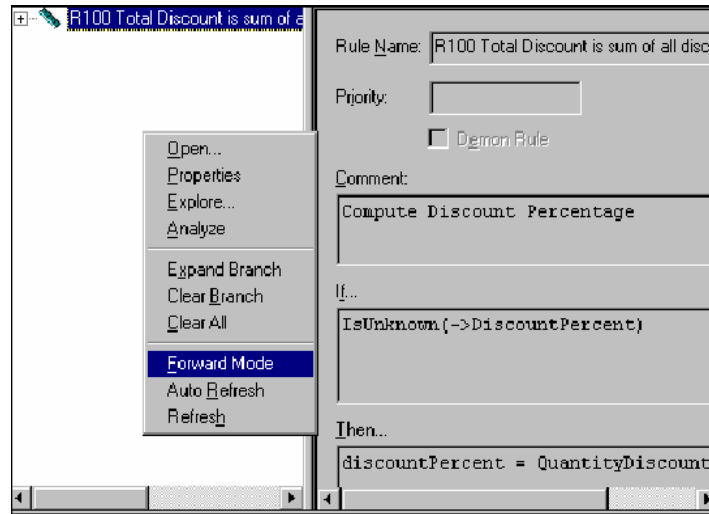
2. Expand the rule method RulesAboutTotalDiscount().
3. Right-click the rule named R100 Total Discount is sum of all discounts. Choose Analyze from the popup menu.

The Rule Analyzer appears in forward mode, which is the default. The mode is listed in the title bar. The left pane shows the attributes needed to resolve the rule, and the right pane contains the Rule Editor.



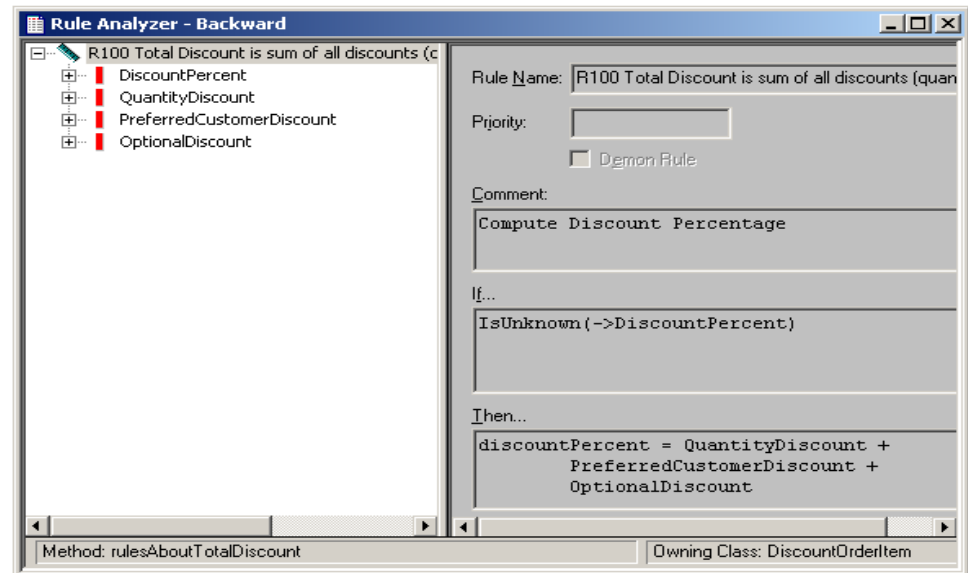
**Note:** The right pane of the Rule Analyzer contains the appropriate editor for the object chosen in the left pane. In this example, the Rule Editor is displayed because a rule was chosen. If an attribute were chosen, the attribute editor would appear.

4. Right-click in the left pane.  
A pop-up menu appears.
5. Choose Forward Mode to clear the check mark.



You have now switched to backward mode.

6. Expand the rule R100 Total Discount is sum of all discounts.



The attributes DiscountPercent, QuantityDiscount, PreferredCustomerDiscount, and OptionalDiscount are listed below the rule, indicating that these attributes are needed to determine the Total Discount.

7. Expand PreferredCustomerDiscount.

The Rule Analyzer displays the rules that determine a value for this attribute.

The screenshot shows the Rule Analyzer interface. On the left, a tree view displays a hierarchy of rules. The root rule is 'R100 Total Discount is sum of all discounts (d...'. It has four children: 'DiscountPercent', 'QuantityDiscount', 'PreferredCustomerDiscount', and 'OptionalDiscount'. The 'PreferredCustomerDiscount' rule is expanded, showing three sub-rules: 'R11 Default Preferred Customer Disc...', 'R05 Preferred Customer in good star...', and 'R06 Preferred Customer with a few I...'. The 'OptionalDiscount' rule is also expanded, showing one sub-rule: 'R07 Preferred Customer without goc...'. On the right, the details for the selected rule 'R100 Total Discount is sum of all discounts (d...' are shown. The 'Rule Name' field contains the rule name. The 'Priority' field is empty. The 'Demon Rule' checkbox is unchecked. The 'Comment' field contains the text 'Compute Discount Percentage'. The 'If...' section contains the condition 'IsUnknown(->DiscountPercent)'. The 'Then...' section contains the action 'discountPercent = QuantityDiscount + PreferredCustomerDiscount + OptionalDiscount'.

Rule Name:	R100 Total Discount is sum of all discounts (d...
Priority:	
<input type="checkbox"/> Demon Rule	
Comment:	Compute Discount Percentage
If...	IsUnknown(->DiscountPercent)
Then...	discountPercent = QuantityDiscount + PreferredCustomerDiscount + OptionalDiscount

8. Close the Rule Analyzer.

Do not save changes.



# Chapter 3: Access to Essential Databases

---

Typically, the information for CA Aion BRE applications is permanently stored in an external file or database. This data can be maintained and accessed by other applications, if necessary, and loaded into CA Aion BRE when needed to solve a particular problem.

This chapter provides information about using databases with CA Aion BRE. In this chapter, you see how a connection to a database is defined in an Aion application, and how to execute the application from the Aion IDE.

## The DataLib System Library

CA Aion BRE simplifies the task of defining the connection to databases. The DataLib library provides the necessary classes for defining this connectivity. Through this connectivity, you can retrieve table definitions at edit time, test SQL statements, and retrieve and update the data stored in a database at runtime. DataLib must be included in any application for which database connectivity is required; it is automatically included whenever you create a new application.

DataLib contains the base classes Connection, Query, and Stored Procedure, as well as the methods necessary for retrieving and updating a database. An application must establish a connection before it can retrieve and update data. To do this, the application uses an instance of the DataLib Connection class. Each different database connection requires its own instance of Connection. In the Discount application, the connection instance is named ConnDiscount.

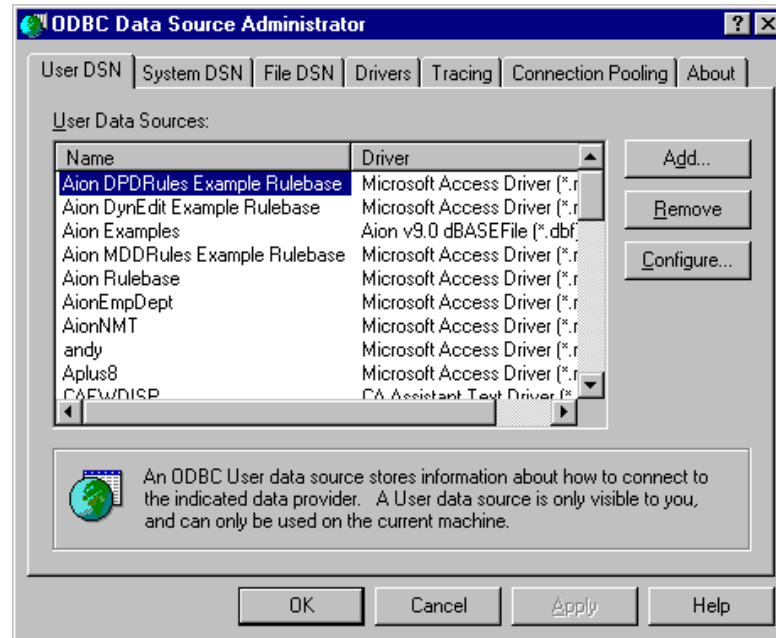
## Specify a Data Source

Before you can run the Discount application, you must define an ODBC data source, using the ODBC Data Source Administrator. A data source specifies the type of database and the location of the database file.

**Note:** The CA Aion BRE setup creates the DiscountDB data source.

### To specify a data source

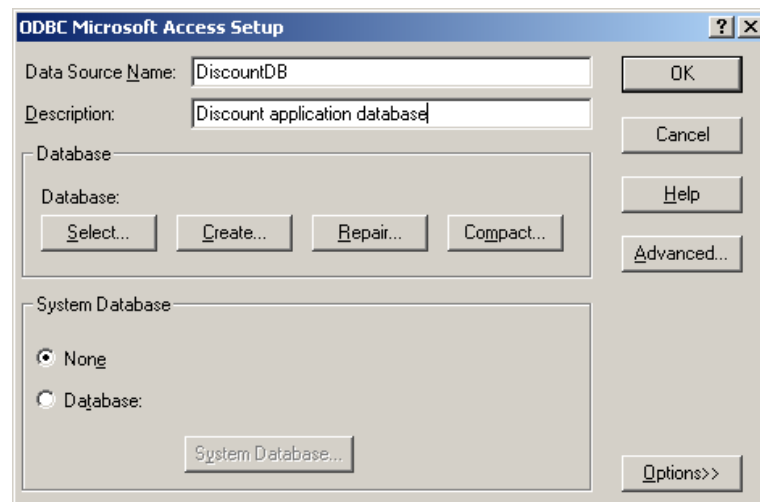
1. From the Windows Control Panel, launch the Windows ODBC Data Source Administrator.



2. Click Add.

The Create New Data Source dialog appears. Select Microsoft Access Driver {\*.mdb} and click Finish.

The ODBC Microsoft Access Setup dialog appears.



3. In the Data Source Name field, enter **DiscountDB**.
4. In the Description field, enter **Discount application database**.
5. Click Select.

The Select Database dialog appears.

6. Navigate to the directory *AionBRE Installation Directory*\examples\Discount\.
7. Select the Discount.mdb database file and click OK.

ODBC creates the DiscountDB data source, which appears on the User DSN page of the ODBC Data Source Administrator dialog.

8. Click OK.

## Test the Data Connection

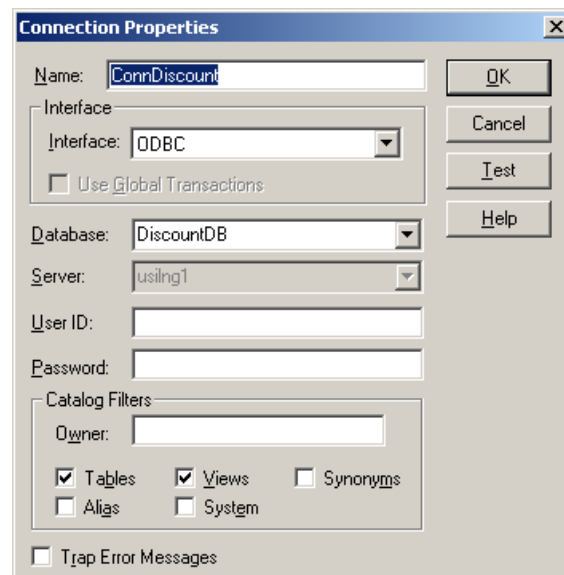
### To verify that CA Aion BRE is connecting to the data source

1. Open BRE IDE from start menu and open discount.app from Examples.
2. Choose Tools, Assistants, Database Assistant.

The Database Assistant dialog appears.

3. Choose the Connection option.
4. Select ConnDiscount and click Next.

The Connection Properties dialog appears



The connection defines how the Discount application accesses the Microsoft Access database.

ConnDiscount uses the ODBC data source that you created named DiscountDB.

5. Click Test to test the connection.

A confirmation dialog appears.

6. Do *one* of the following:

- If the connection is successful, continue with step 6.
- If the connection is unsuccessful, repeat the "Specify a Data Source" procedure, then test the connection again.

7. Click OK to close the confirmation dialog.

8. Click OK.

## Create a Query Class

After a connection is successfully established, you can use the Query Editor to define a query class and perform the following tasks:

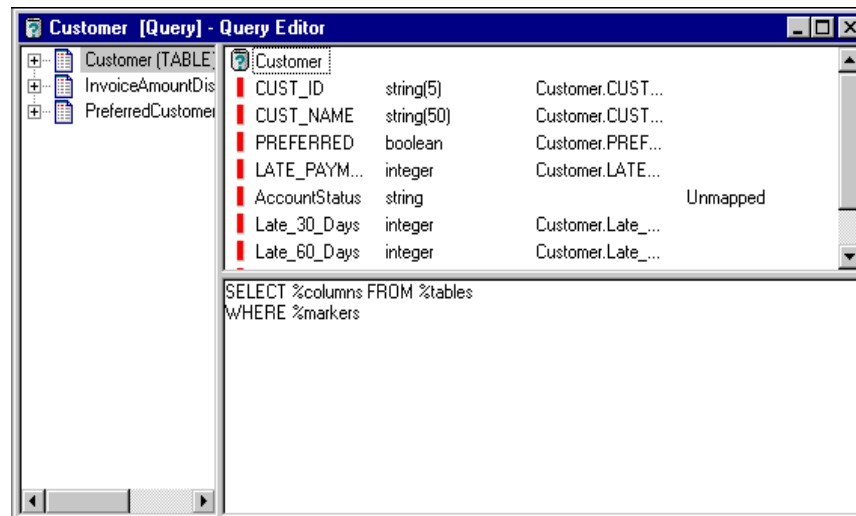
- Browse through database tables and the columns within those tables
- Define field attributes for the table columns
- Change the SELECT statement of the query



**To use the Query Editor**

1. In the Project Workspace, select the Classes tab.
2. Expand Menus and Tools.
3. Expand Queries and Stored Procedures.
4. Under Queries and Stored Procedures, double-click the query class Customer.

CA Aion BRE connects to the Discount database through the DiscountDB data source, and the Query Editor appears.

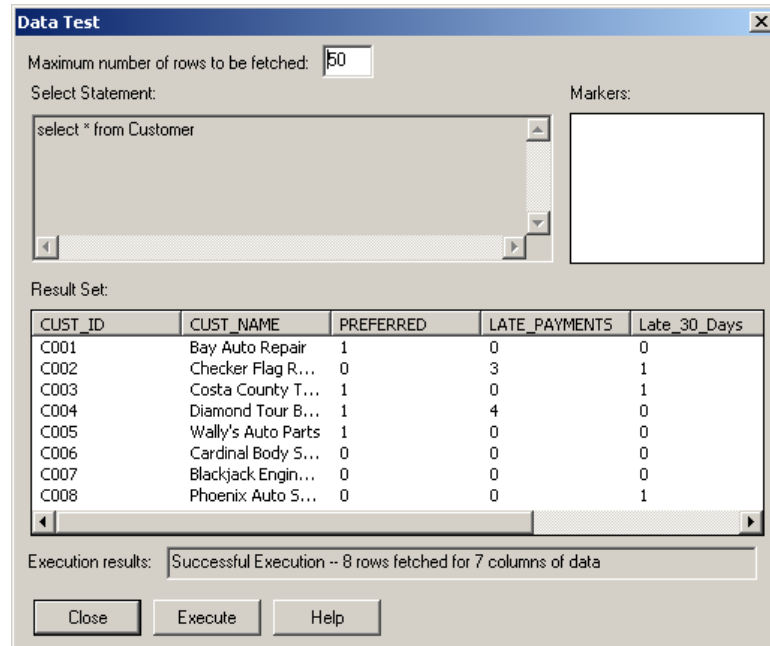


5. In the left pane, right-click the Customer table.

A pop-up menu appears.

6. Choose Open.

CA Aion BRE sends a default query to the database, and retrieves up to the first 50 records in the table.



7. Click Close to close the Data Test dialog.

8. Close the Query Editor.

If you are prompted to save the changes, click No.

## Execute the Discount Application in the Aion IDE

CA Aion BRE provides you with two ways to execute an application - you can compile the application into an executable, or you can run the application in interpreted mode. Interpreted mode is ideal for iterative development because you can make incremental changes to the code and then test the application by executing it quickly or repeatedly without taking the time to compile it. Compiled execution is typically used to provide high performance execution of the application in a production environment. You will learn to compile the application in the chapter [Build and Deploy Applications and Components](#) (see page 71).

**To run the Discount application in interpreted mode**

1. Choose File, Run, and then Run.

After a few seconds, the database connection is established and the Discount application window appears. The application name is Redwood Auto Supply Pricing Application.

**Redwood Auto Supply Pricing Application**

File Edit

Customer: [Drop-down menu]  
☐ Preferred Customer  
 Acct Status: ?



Season:  
☐ Spring  
☐ Summer  
☒ Autumn  
☐ Winter

Resolve Discounts  
 Place Order

Quantity: 0 Product: [Drop-down menu] [Blue Arrow Button]

Qty	Product	List Price	Discount

Discount Explanation

2. From the Customer drop-down menu, select Wally's Auto Parts.  
 The Preferred Customer check box automatically selects and the Acct Status becomes Good. These status items are dynamically calculated by rules that evaluate the customer history in the database.
3. From the Product drop-down menu, select Power Steering Fluid.
4. In the Quantity field, enter **11** then click  to add this item to the order list.  
 The item Power Steering Fluid appears in the order list.
5. From the Product drop-down menu, select Intake Manifold then click  to add this item (with the same order quantity as the Power Steering Fluid) to the order list.
6. The item Intake Manifold appears in the order list.

7. Click the Resolve Discounts button.

The application calculates the various discounts that apply and displays the total discount for each product.

**Redwood Auto Supply Pricing Application**

File Edit

**Customer**  
Wally's Auto Parts  
☒ Preferred Customer  
Acct Status: Good

**Season**  
☐ Spring  
☐ Summer  
☒ Autumn  
☐ Winter

**Resolve Discounts**  
**Place Order**

**Quantity** **Product**  
11 Intake manifold

Qty	Product	List Price	Discount
11	Power steering fluid	\$120.45	27 %
11	Intake manifold	\$75.35	14 %

**Discount Explanation**  
SeasonalDiscount Decision Table for Autumn is 4%  
R05 Preferred Customer in good standing gets 10% discount  
R03 Discount for quantities under 50 is 0%  
R10 Discount for product Power steering fluid is 13%  
R100 Total Discount is sum of all discounts [quantity, seasonal, product, etc.]

8. Select an item in the order.

The Discount Explanation area lists the rules executed for each line item, thus providing a justification for the total discount.

Play with the application, changing its various inputs. What happens if you toggle the Preferred Customer box so that it is unchecked but leave the season set to Autumn? In particular, why is there no more SeasonalDiscount recognized? (Look back at the SeasonalDiscountTab rule method, the one that consists of a decision table. What is the meaning of the highlighted empty cell?)

For an extra challenge, see if you can fix this problem and rerun the application so that a season discount is effect for a non-preferred customer in Autumn?

### More Information:



[Build and Deploy Applications and Components](#) (see page 71)

## Execute the Discount Application in the Aion IDE

### To run the Discount application in interpreted mode

1. Choose File, Run, and then Run.

After a few seconds, the database connection is established and the Discount application window appears. The application name is Redwood Auto Supply Pricing Application.

2. From the Customer drop-down menu, select Wally's Auto Parts.  
The Preferred Customer check box automatically selects and the Acct Status becomes Good. These status items are dynamically calculated by rules that evaluate the customer history in the database.
3. From the Product drop-down menu, select Power Steering Fluid.
4. In the Quantity field, enter **11** then click  to add this item to the order list.  
The item Power Steering Fluid appears in the order list.
5. From the Product drop-down menu, select Intake Manifold then click  to add this item (with the same order quantity as the Power Steering Fluid) to the order list.
6. The item Intake Manifold appears in the order list.

7. Click the Resolve Discounts button.

The application calculates the various discounts that apply and displays the total discount for each product.

**Redwood Auto Supply Pricing Application**

File Edit

**Customer**  
Wally's Auto Parts  
☒ Preferred Customer  
Acct Status: Good

**Season**  
☐ Spring  
☐ Summer  
☒ Autumn  
☐ Winter

**Resolve Discounts**  
**Place Order**

**Quantity** **Product**  
11 Intake manifold

Qty	Product	List Price	Discount
11	Power steering fluid	\$120.45	27 %
11	Intake manifold	\$75.35	14 %

**Discount Explanation**  
SeasonalDiscount Decision Table for Autumn is 4%  
R05 Preferred Customer in good standing gets 10% discount  
R03 Discount for quantities under 50 is 0%  
R10 Discount for product Power steering fluid is 13%  
R100 Total Discount is sum of all discounts [quantity, seasonal, product, etc.]

8. Select an item in the order.

The Discount Explanation area lists the rules executed for each line item, thus providing a justification for the total discount.

Play with the application, changing its various inputs. What happens if you toggle the Preferred Customer box so that it is unchecked but leave the season set to Autumn? In particular, why is there no more SeasonalDiscount recognized? (Look back at the SeasonalDiscountTab rule method, the one that consists of a decision table. What is the meaning of the highlighted empty cell?)

For an extra challenge, see if you can fix this problem and rerun the application so that a season discount is effect for a non-preferred customer in Autumn?

# Chapter 4: Develop an Aion BRE Knowledge Base

---

This chapter demonstrates the logistics of building a simple Aion BRE knowledge base. In this chapter, use the material you have learned so far to construct a simple Aion application.

## The Purchase Checker System

The following tours concentrate on database connectivity (loading and processing data from a database) and on using the inferencing capabilities of CA Aion BRE.

### System Specification

A client asks you to develop a system that checks customer purchases to determine if the total price falls within the allowed credit limit for the customers. This system will run against a set of customers in a database, so the system will have to read customer records, process those records against rules for accepting purchases, and report the results. The name of the system is the Purchase Checker System.

### Requirements

The system should approve a customer purchase if the total credit amount is less than or equal to a stipulated credit limit that is based on marital status and annual income. Otherwise, the purchase is not approved. You are given the following rules for determining the allowable credit limit:

- For customers with annual income less than or equal to \$20,000, a single person has a credit limit of \$500 and a married person has a credit limit of \$400.
- For customers with annual incomes greater than \$20,000 but less than or equal to \$55,000, a single person has a credit limit of \$1,200 and a married person has a credit limit of \$1,000.
- For customers with annual incomes greater than \$55,000, a single person has a credit limit of \$5,000 and a married person has a credit limit of \$4,000.

The customer credit limit is to be included in a summary evaluation report.

## Design Assumptions

You believe that the application will make an elegant, object-oriented knowledge base. Your thoughts for designing it are as follows:

- You realize that the allowable credit limit rules are ideal for a decision table. Your client is receptive to the idea of representing these rules as a decision table.
- You envision a Customer class with data derived from the database and supplemented with methods that make it an intelligent object. The Customer class will have methods that evaluate the credit worthiness of each customer and report the result of this evaluation.
- You also envision a Purchase Checker Agent, an abstract class that “knows” how to invoke the methods of the Customer class to evaluate a particular customer. This Agent class controls the problem-solving strategy of the knowledge base.

## Build the Purchase Checker System

CA Aion BRE provides many ways to code a system design. Some approaches are bottom-up—they start with the basic elements of the design and work upward to the more general elements. Other approaches are top-down—they start with the general functionality and work down to the basic elements.

Because the Purchase Checker System relies heavily on database connectivity, you will begin with a bottom-up approach.

**Note:** In developing a knowledge base, you would normally test functionality frequently, as soon as a small unit of work is coded. A bottom-up approach does not allow rapid testing. Therefore, as soon as you have sufficient functionality coded to test, you will switch to the top-down method.

To begin creating your Purchase Checker application, you will initialize it and then make a connection to the database. You will then create a query class to view the instances in the database, map that query class to the database, and test the database connection. Finally, the database will be viewed in the Aion Debugger.

**Note:** A completed version of this version is available in *AionBRE Installation Directory*\examples\purcheck\purcheck.app.



## Create the Application

### To create a knowledge base

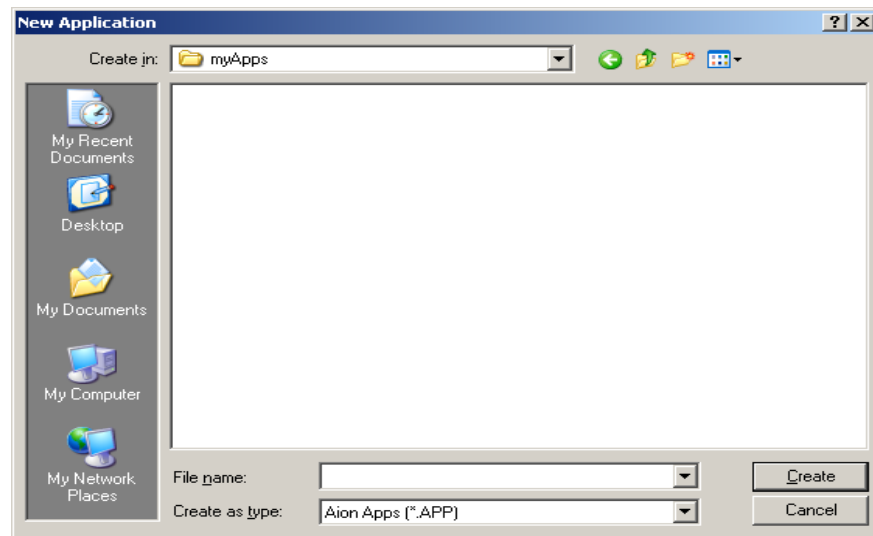
1. Choose Tools, Assistants, Application Assistant.

The Application Assistant dialog appears.

2. Click Next.

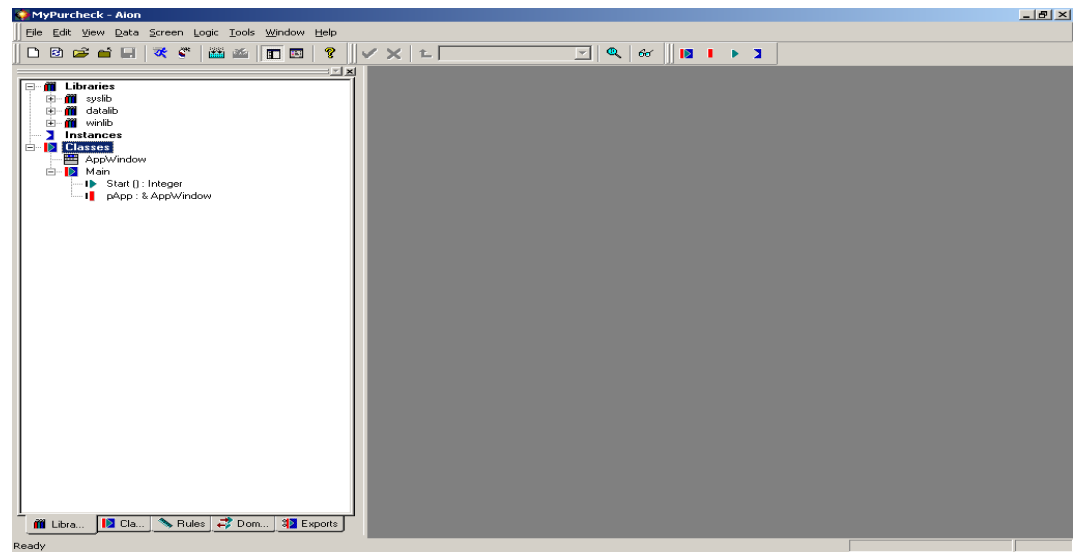
The New Application dialog appears.

3. Click the Create New Folder icon  and create a parent directory for your new knowledge base called myApps.



4. In the File name field, enter a name for the CA Aion application; for example, enter **MyPurchase**.
5. Click Create.

In the Project Workspace, default libraries and classes appear.



6. In the Application Assistant dialog, select the Standalone Executable option and click Next.
7. Accept the default system libraries by clicking Next.
8. Select the Database option and click Done.

The Database Assistant dialog appears.

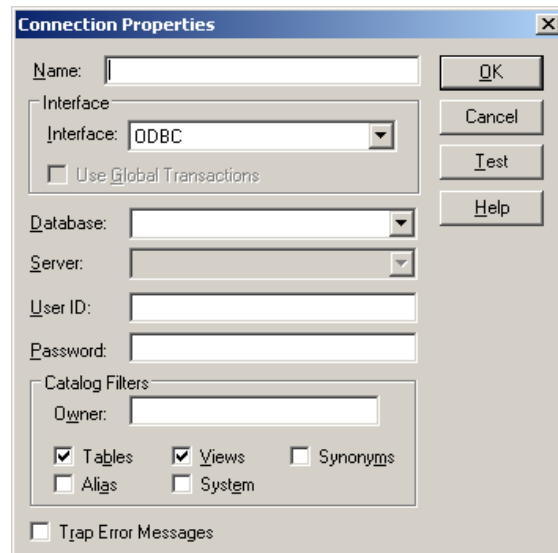
## Connect to the Database

The database for this tutorial is provided in *AionBRE Installation Directory*\examples\purcheck\Customer.dbf. The following procedure shows how to make a connection to that database and test the connection. To connect to the database, use the Database Assistant to help you set up the connection.

### To connect to the database

1. If the Database Assistant dialog is open from the previous tour segment, continue with the next step. If the Database Assistant is not open, choose Tools, Assistants, Database Assistant.
2. Select the Connection option and <New>. Click Next.

The Connection Properties dialog appears.



3. In the Name field, enter **Cust\_Connection**.
4. From the Database drop-down menu, choose Aion Examples.
5. To test the connection, click Test.

A message appears confirming that the connection was successful.

6. Click OK to close the message box.
7. Click OK to close the Connection Properties dialog.
8. On the Database Assistant dialog, click Done.

## Create a Query Class

A query class allows you to view the data in a database. The following procedure shows how to create a query class that lets you see the Customer database that is provided for this tour. To create a query class, use the Database Assistant.

### To create a query class

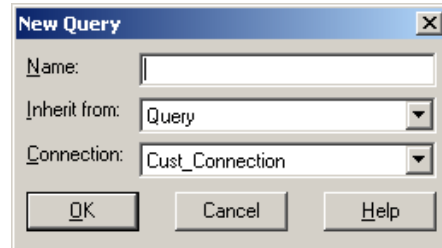
1. Choose Tools, Assistants, Database Assistant.

The Database Assistant dialog appears.

2. Select the Query option and click Next.

The New Query dialog appears.

*Equation 1: Show screen to create a new query class*

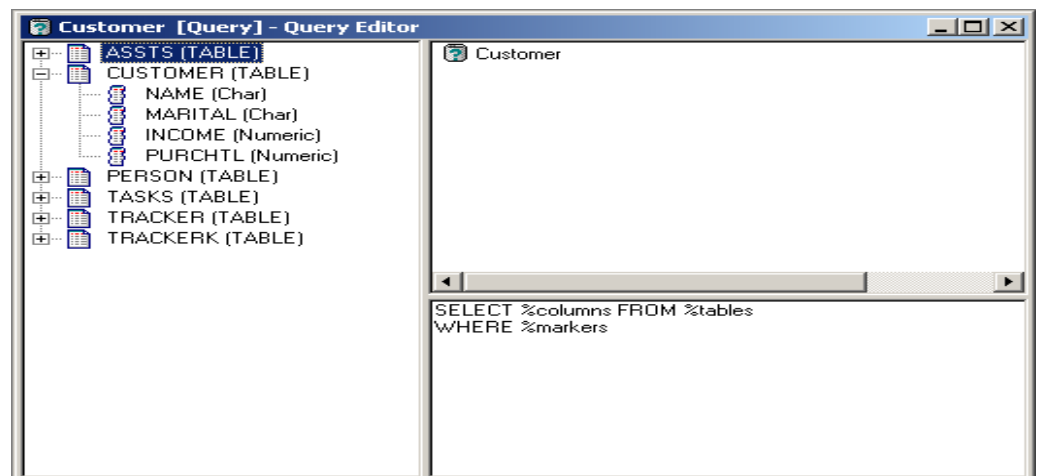


The Inherit from field contains Query and the Connection field contains Cust\_Connection.

3. In the Name field, enter **Customer** as the name of the class and click OK.

The Query Editor opens in the Main window. The left pane lists all the sample databases in the directory *AionBRE Installation Directory\examples\purcheck*. For the purposes of this tour, you will use the Customer table.

4. On the Database Assistant dialog, click Done.
5. Expand the Customer table to view its columns (fields).



## Map the Query Class to the Database

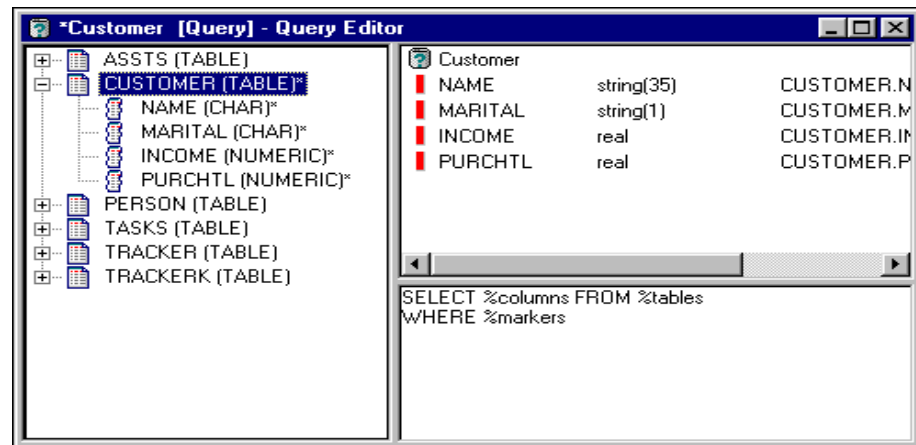
### To map the Customer query class to the Customer table

1. In the left pane of the Query Editor, right-click CUSTOMER (TABLE).

A pop-up menu appears.

2. Choose Use Column.

The top right-hand pane fills with column information. The Customer table columns are now mapped to the public attributes of the Customer Query class.



You may want to change the abbreviated, capitalized names of these attributes.

3. In the top right-hand panel, double-click NAME.

The Properties dialog appears.

4. Change NAME to Name and click OK. Repeat steps 3 and 4 to change MARITAL to MaritalStatus, INCOME to Income, and PURCHTL to PurchaseTotal.
5. Click the Save icon on the toolbar.
6. Click Query Editor Save "check mark icon" to save the changes and close the Query Editor.

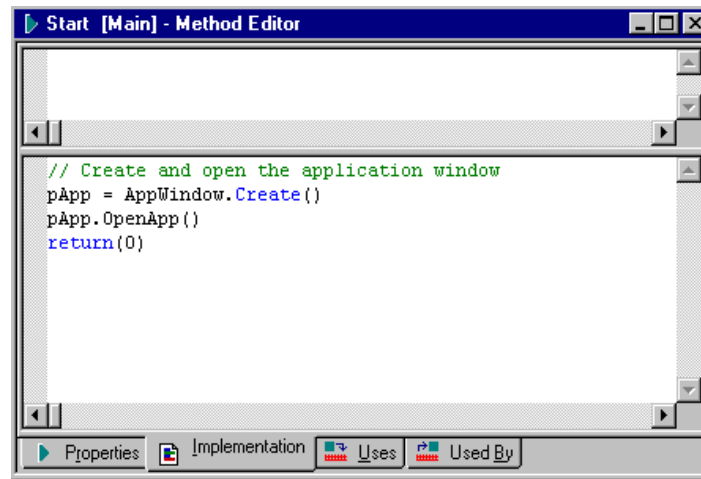
## Test the Database Connection

Now that you have coded a small part of the application, it is time to test what you have done. In this section, you switch to a top-down approach and enter code to start the application.

### To write code to test the database connection

1. On the Libraries page of the Project Workspace, expand Classes.
2. Expand the Main class.
3. Double-click the Start() method.

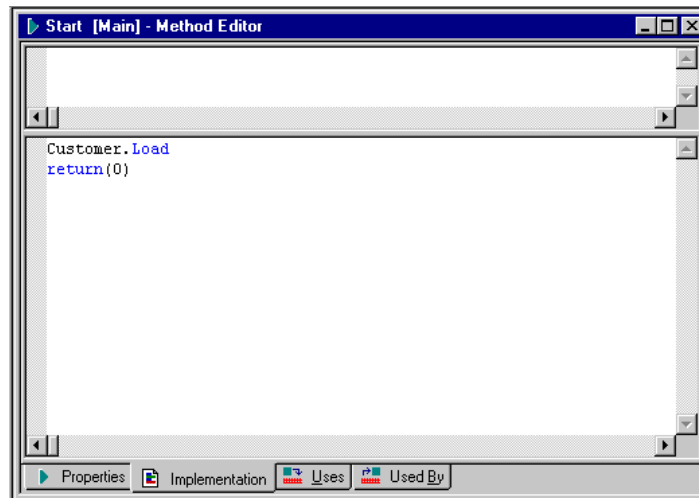
The Method Editor appears in the Main window and shows the code in the Start() method.



The Start() method is the first method to execute in every Aion application. This method provides code to create an application window. Because you are not going to use an application window in this example, you must change the code.

4. To edit the code, follow these steps:
  - a. Delete the comment line and the first two lines of code that create and open the application window, but leave the return statement. The return statement is required to be the last line of code of the Start() method.
  - b. As the first line to be executed, enter the following code, which loads the database:  

```
Customer.Load
```
  - c. Ensure that your code appears as shown in the following graphic:




5. On the toolbar, click the Save icon.
6. Save Method Editor changes by clicking "check mark icon" to save the changes and close the Method Editor.

## View the Database in the Aion Debugger

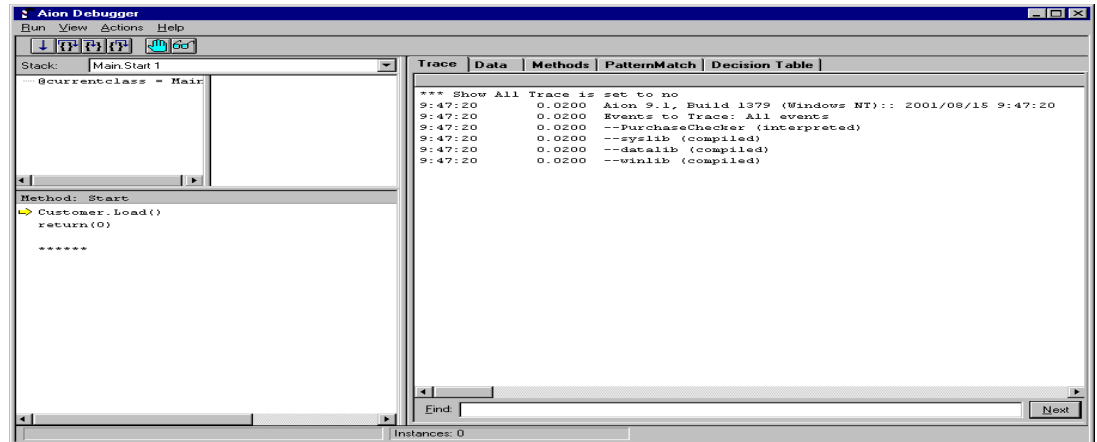
To ensure that the Customer query class loads the Customer database correctly, run the Aion Debugger.


**Note:** For more information about the Aion Debugger, see the appendix "The Aion BRE Debugger" and the *CA Aion BRE Product Guide*.

## To run and use the Aion Debugger

1. Click the debugger icon .

The Aion Debugger window appears.

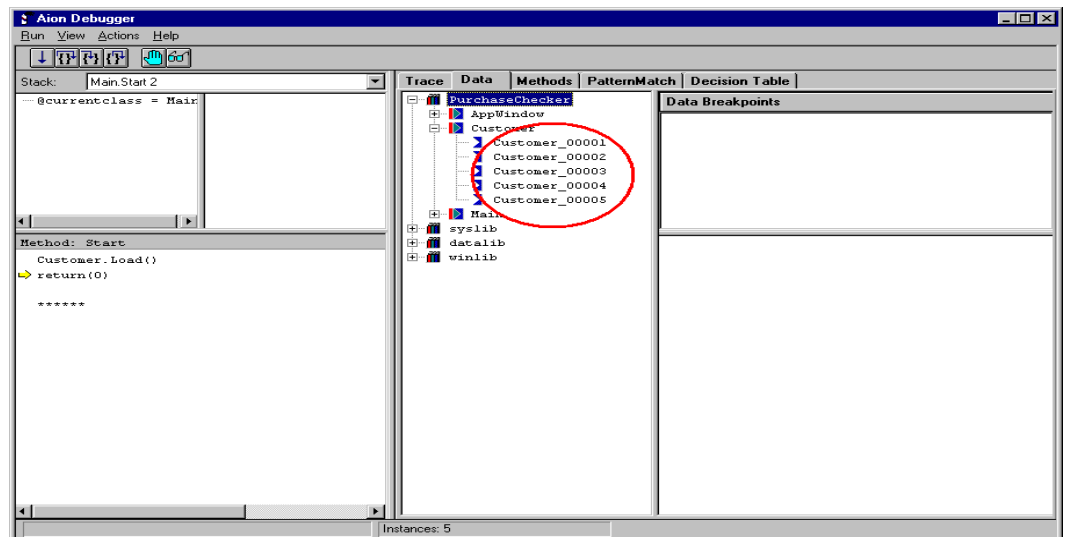


2. On the toolbar, click the Step Over button .

The Customer.Load() method loads the instances from the Customer database.

3. In the right-hand pane, click the Data tab. Expand the MyPurchase library, and then expand the Customer class.

Notice that five instances were loaded from the database.



4. Click an instance, for example Customer\_00001.

The instance value attributes appear in the lower right-hand pane.

5. Press F5 to close the Aion Debugger.




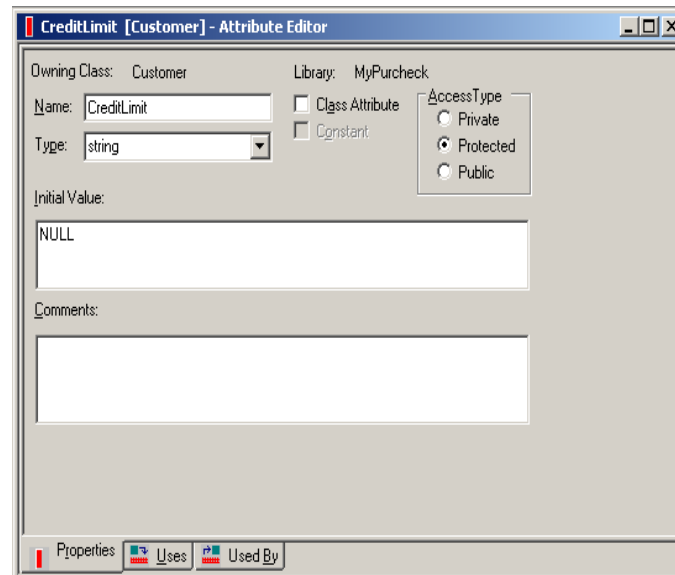
## Create Attributes to Store Credit Limit and Approval

Now you return to the bottom-up approach and create two attributes in the Customer class:

- **CreditLimit**, which, as determined by the rules, holds the credit limit for the customers
- **Approval**, which holds the final purchase approval determination

### To create new attributes

1. In the Project Workspace, select the Customer class.
2. On the toolbar, click the Attribute icon  .  
The New Attribute dialog appears.
3. In the Name field, enter **CreditLimit**.
4. From the Owning Class drop-down menu, choose Customer, and click OK.  
The Attribute Editor appears.




5. In the Attribute Editor, for Access Type, choose the Protected option.
6. From the Type drop-down menu, choose real.
7. Remove NULL from the Initial Value field. This attribute is used in inferencing.
8. Close the Attribute Editor.
9. Repeat steps 1 through 8 to define the Approval attribute with a Type of boolean.
10. Save Changes in Current Editor dialog appears, reply Yes.

## Create a Method to Calculate Credit Limit

Because the application will set credit limits for purchases, you must create a `SetCreditLimit()` method, which a decision table (that you will also create) uses. You designate the `SetCreditLimit()` method as a domain interface method. Get and Set accessor methods, like this one, are often used as domain interface members.

**Note:** You can also create accessor methods (Get and Set) automatically. Highlight the attribute for which accessors are to be created and select Create Accessors from the Logic menu. (It is usually sufficient to accept the defaults that appear on the Create Accessors dialog.)

### To create the `SetCreditLimit()` method

1. Select the Customer query class and click the Method icon  on the toolbar.  
The New Method dialog appears.
2. Complete the following fields, and click OK:

#### **Name**

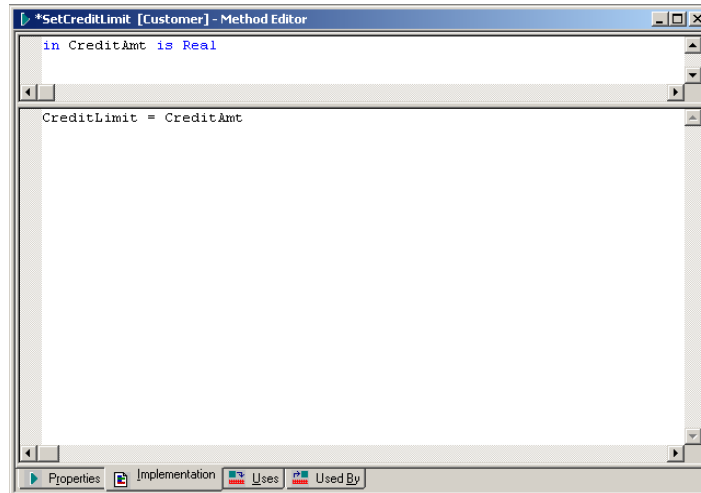
Enter `SetCreditLimit`.

#### **Owning Class**

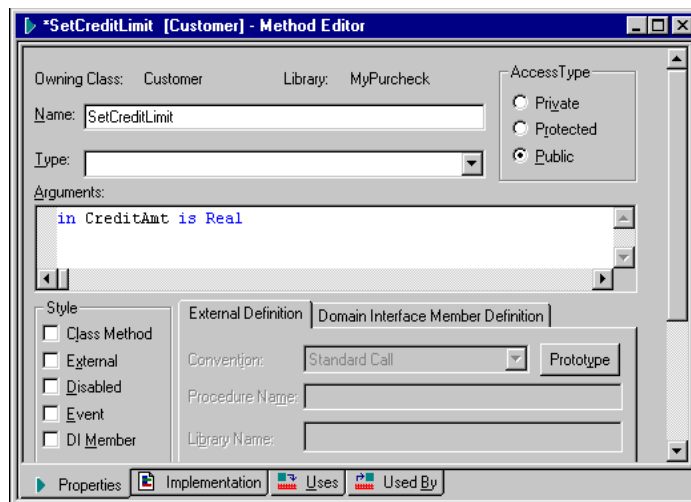
Choose Customer from the drop-down menu.

3. Select the Implementation tab.
4. To define an input argument called CreditAmt with a type of Real, in the top pane, enter the following code:  

```
in CreditAmt is Real
```
5. Ensure that your code appears as shown in the screen below:



6. Select the Properties tab.



7. Complete the fields:

#### Access Type

Select the Public option.

#### Style

Select the DI Member check box.

8. Select the Domain Interface Member Definition tab.

9. Complete the following fields:

**Type**

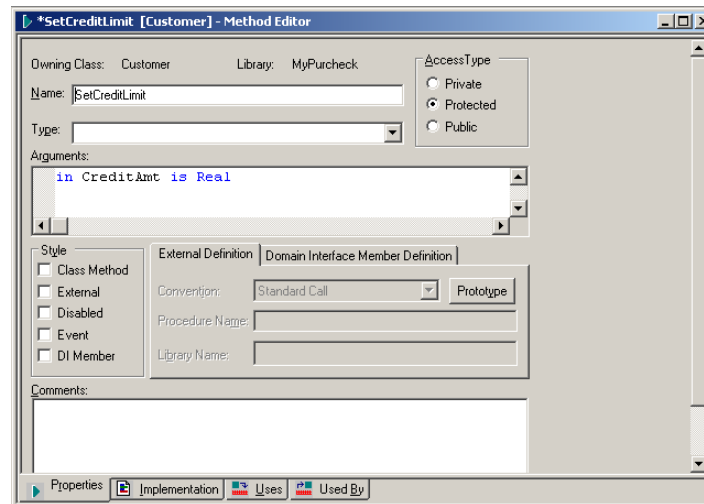
Select the Action option.

**Label**

Enter Allowable credit is =.

**Description**

Enter Sets the credit amount or total purchase value that is authorized for a customer.



10. Click the "check mark icon" on the Method editor panel to save the changes, and close the Method Editor.

## Create a Decision Table

You will now define your first rule, which is a decision table for setting the CreditLimit attribute.

After you complete your decision table, it appears as shown below:

MaritalStatus	Annual Income	Allowable credit is =
"S"	<=20000	500
	>2000..<=55000	1200
	>55000	5000
"M"	<=20000	400
	>2000..<=55000	1000
	>55000	4000

**To create a decision table**

1. Select the Customer query class.
2. Choose Logic, New, Decision Table.  
The New Decision Table dialog appears.
3. Complete the fields:

**Name**

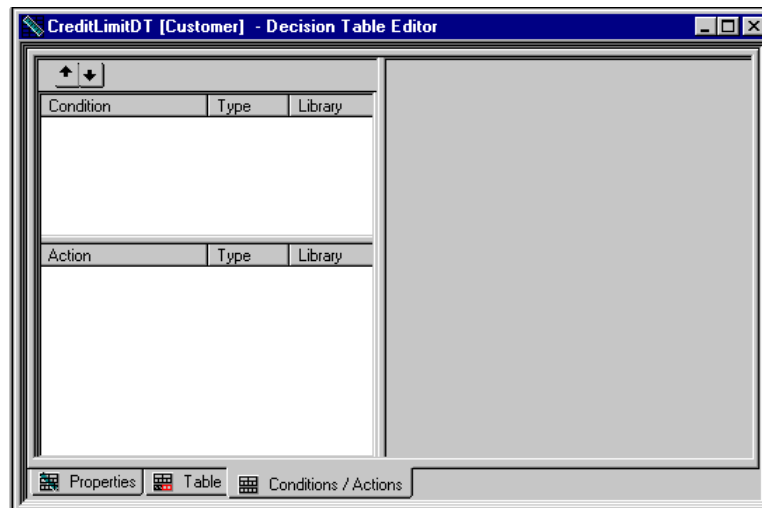
Enter CreditLimitDT.

**Owning Class**

From the drop-down menu, choose Customer.

4. Click OK.

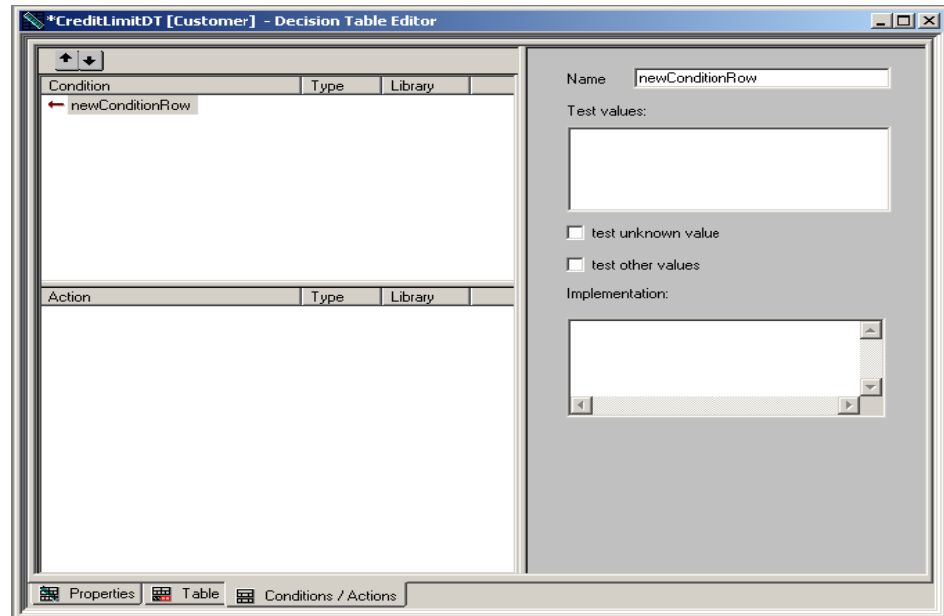
The Decision Table Editor appears.



5. Select the Conditions/Actions tab.  
You now define the first condition.
6. Right-click in the Condition/Type/Library pane.  
A pop-up menu appears.

7. Choose New, Specify Condition.

The right-hand pane fills with empty fields.



8. Complete the fields:

**Name**

Enter Marital Status.

**Test Values**

Enter **"S"**, click Enter, and enter **"M"**. Quotation marks are required.

**Implementation**

Enter **MaritalStatus**, which associates the new condition with the existing MaritalStatus public attribute. Repeat steps 6-8, completing the fields as follows:

**Name**

Enter Annual Income.

**Test Values**

Enter:

<=20000

>20000 . . <=55000

>55000

**Implementation**

Enter Income.

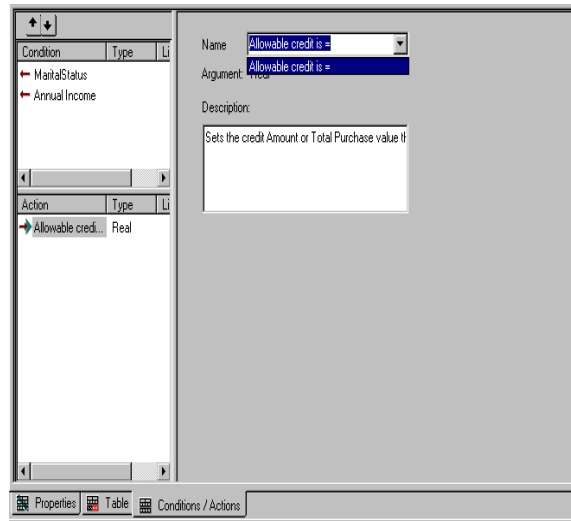
Next you need to define an action.

9. Right-click in the Action/Type/Library pane.

A pop-up menu appears. Because you want to pass the credit limit to the interface as an input argument, the action is domain.

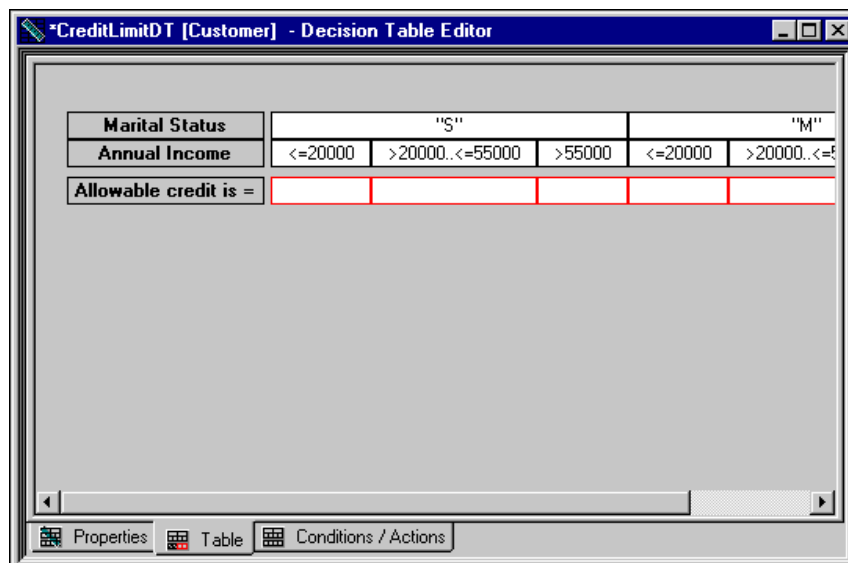
10. Choose New, Domain Action.

11. From the Name drop-down menu, choose Allowable credit is =.



12. Select the Table tab.

The decision table, with the specified conditions, appears.



**Note:** If the decision table is not automatically created, right-click on the empty page and, from the pop-up menu, choose Refresh or Auto Refresh.

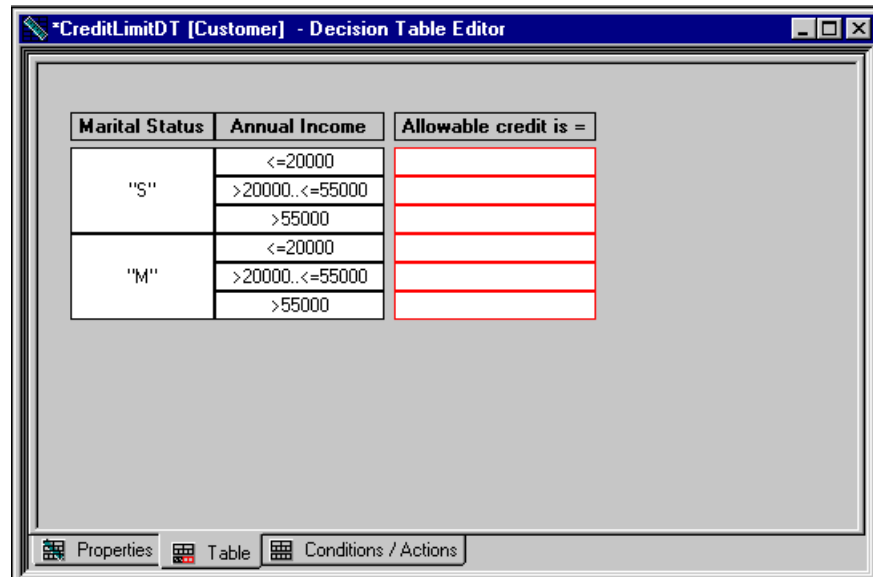
Because the decision table will be easier to work with in portrait format, toggle the layout.

13. Right-click anywhere outside of the table.

A pop-up menu appears.

14. Choose Toggle Layout.

The decision table updates in portrait layout.



15. For the Allowable credit is = cells, enter the following values: **500, 1200, 5000, 400, 1000, 4000**.

16. Ensure that your table appears as shown below:

MaritalStatus	Annual Income	Allowable credit is =
"S"	<=20000	500
	>2000..<=55000	1200
	>55000	5000
"M"	<=20000	400
	>2000..<=55000	1000
	>55000	4000



17. Click the Save Icon "Check Mark" on the Decision Table Editor, and close the Decision Table Editor.

Your new decision table is a static decision table. You might have chosen instead to create a dynamic decision table, which could be defined and stored in a rulebase outside CA Aion BRE. However, in order for CreditLimitDT to be a dynamic decision table, you must define accessor methods (Get methods) in the application for the MaritalStatus and Income attributes. You would designate the accessor methods as domain interface members, as you did for SetCreditLimit(). You would export the domain interface members to a rulebase.

For more information on dynamic rules, see the *CA Aion BRE Rules Guide*.

## Post the Decision Table

You now define an inference method to post and execute your new decision table. This inference method is invoked by the Purchase Checker Agent class (which you will create also). You know from the design that this class will contain the control strategy of the application.

### To post the decision table

1. Select the Customer query class.
2. Choose Logic, New, Method.

The New Method dialog appears.

3. Complete the fields:

#### **Name**

Enter DeterminePurchaseLimit.

#### **Owning Class**

From the drop-down menu, choose Customer.

4. Click OK.  
The Method Editor Appears.
5. Select the Properties tab.
6. For the Access Type, select the Public option.
7. Select the Implementation tab. In the lower pane, enter the following code:

```
INFER
CreditLimitDT      //posts the decision table
ForwardChain        //invokes inferencing to
                    //execute the decision table

END
```


The implementation for this method is an inference block that posts the decision table and invokes forward chaining. Either backward chaining or forward chaining is appropriate for this application. In general, the default should be forward chaining.

8. Click the Save Icon "check Mark" on the Method Editor, and close the Method Editor.

## Call the Decision Table

In this section, you create a class and a method within that class. The method contains a loop that calls the CreditLimitDT decision table for each instance in the Customer database.

**To call the decision table**

1. On the toolbar, click the Class  icon.

The New Class dialog appears.

2. Complete the fields:

**Name**

Enter PurchaseCheckAgent.

**Base Class**

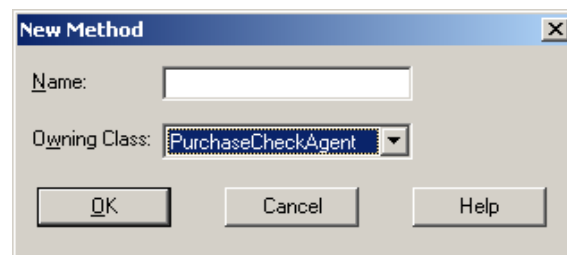
From the drop-down menu, choose \_System.

3. Click OK.

The Class Editor appears.

4. Choose Logic, New, Method.

The New Method dialog appears.



5. Complete the fields:

**Name**

Enter CheckCustomer.

**Owning Class**

From the drop-down menu, choose PurchaseCheckAgent.

6. Click OK.
7. Select the Properties tab. Ensure that the fields appear as follows:

**Style**

Class Method (which it should be automatically because you chose \_System as the base class)

**Access Type**

Public

**Note:** The Access Type must be Public because the Main.Start() method calls it.

8. Select the Implementation tab. In the lower pane, enter the following code:

```
Var pCust is pointer to Customer
For Customer, pCust
    PCust.DeterminePurchaseLimit()
End
```

This code creates a loop that calls the DeterminePurchaseLimit() method for each instance loaded from the Customer database.

9. Click the save Icon "check mark" in the Method Editor pane, and then the Class Editor and on the toolbar.

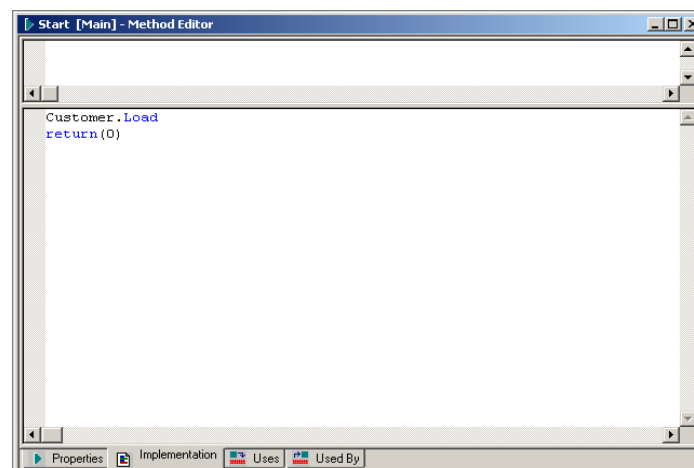
## Test the Decision Table

You should check the new functionality by going back to the top-down approach. In this section, you add a statement to the Start() method, which you edited earlier in Test the Database Connection in the chapter "Access to Essential Databases."

### To test the decision table

1. Expand the Main class, then double-click the Start() method.

The Method Editor appears.




2. Select the Implementation tab. In the lower pane, modify the code as follows:

```
Customer.Load()
CheckCustomer()
return(0)
```

This code invokes the CheckCustomer() method, which is a public class method and is therefore globally accessible.

3. Close the Method Editor and save the application by clicking Yes.

4. Click the Debug  button.

The Aion Debugger dialog appears. Click the Step Over  button.

This button executes the Customer.Load() method, which loads instances from the Customer database. Notice that the yellow arrow in the lower left pane points to the next method in Start(), which is CheckCustomer().

```
Method: Start
Customer.Load
➔ CheckCustomer()
return()

*****
```

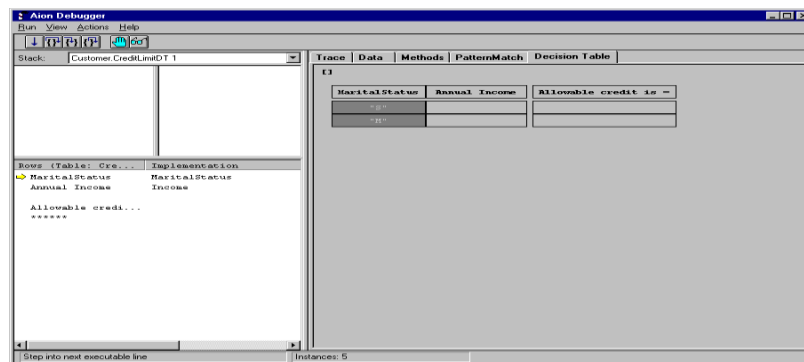
You will now view and execute the code in the CheckCustomer() method.

5. Click the Step Into  button.

```
Method: CheckCustomer
➔ var pCust is pointer to Customer
  For Customer, pCust
    pCust.DeterminePurchaseLimit( )
  End

*****
```

6. Click Step Into several times to execute each line in the CheckCustomer() method and in the method with the inference block, DeterminePurchaseLimit().
7. Before the CreditLimitDT method is executed, select the Decision Table tab.



The Decision Table page is blank until a decision table is posted to the inference engine or a decision table is actually being executed by the inference engine. When the inference engine is executing the decision table, the Decision Table page shows the execution of each instance through the decision table itself.

8. After the loop in `CheckCustomer()` has completed, check the Customer instances to see the credit limit assigned to each one.
9. Press F5 to close the Aion Debugger.

## Rules to Approve Purchases

Now you must write the final rules for approving a purchase based on the credit limit that was just assigned to the customer. Two new rules are needed: a rule for purchase approval and a rule for purchase disapproval. After writing these rules, you will test them in the debugger.

### Write the Approval Rule

The rule for approval is simple: if the total purchase is less than or equal to the credit limit, the purchase is approved. One way to write this rule is to use the Rule Editor.

#### To write the approval rule

1. Select the Customer query class.
2. Choose Logic, New, Rule Method.  
The New Rule Method dialog appears.
3. Complete the fields:

##### **Name**

Enter ApprovalRule.

##### **Owning Class**

Choose Customer from the drop-down menu and click OK.

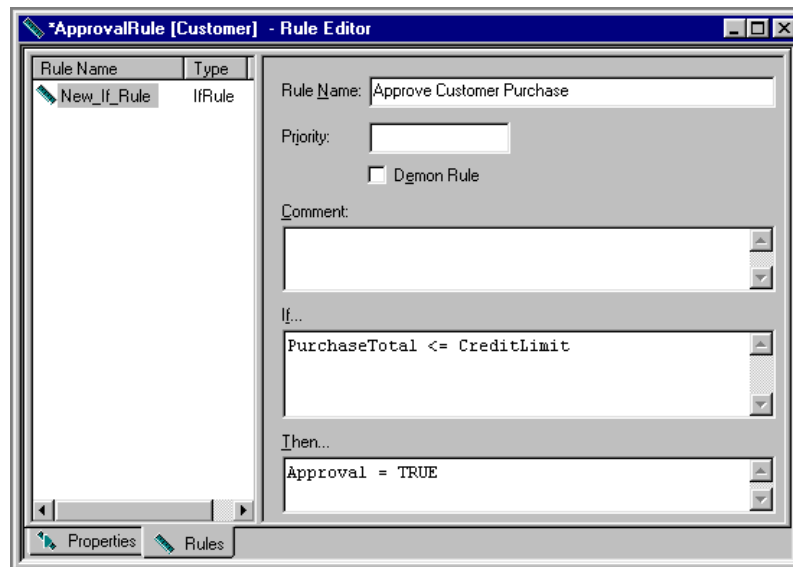
The Rule Editor appears. By default, rule methods have a Protected Access Type.

4. Select the Rules tab.
5. In the left pane, right-click.

A pop-up menu appears

6. Choose New, If Rule.

The new If Rule fields display in the right pane.



7. Complete the fields:

**Rule Name**

Enter Approve Customer Purchase.

**If...**

Enter PurchaseTotal <= CreditLimit.

**Then...**

Enter Approval = TRUE.

8. Click Save Icon "check mark" on the Rule Editor.
9. Click the Save icon on the toolbar, and then close the Rule Editor.

## Write the Disapproval Rule

The rule for disapproval is a “default” rule; that is, if ApprovalRule does not fire, the Approval attribute is set to FALSE. To express this in CA Aion BRE, write a rule that tests whether the Approval attribute is still Unknown after the Approval rule has been processed. CA Aion BRE provides a method for testing whether an attribute is unknown: IsUnknown(), which takes a pointer to an attribute as an input argument.

In this section, you create the disapproval rule using the Method Editor rather than the Rule Editor. Rule methods are just standard methods except that they contain declarative code.

### To create the disapproval rule

1. Select the Customer query class.
2. Choose Logic, New, Method.

The New Method dialog appears.

3. For the Name field, enter **DisapprovalRule**, and click OK.
4. In the Method Editor, select the Implementation tab. In the lower pane, enter the following code:

```
RULE "Disapprove Customer Purchase"
PRIORITY -5
IFRULE IsUnknown(->Approval)
THEN Approval = FALSE
END
```

**Note:** To ensure that this rule is the last rule that the inference engine considers, you have specified a negative priority. You will see another way to ensure that this rule is considered last when you write the inference method to post the two approval rules.

5. Select the Properties tab.
6. For the Access Type, select the Public option.
7. Click the save icon "Check mark" on the Method Editor
8. Click the Save icon on the toolbar, and close the Method Editor.



## Post the Rules

Having written rule methods, you must now write the inference method that invokes them.

### To post the rules

1. Select the Customer query class.
2. Choose Logic, New, Method.  
The new method dialog appears.
3. For the Name field, enter **DetermineApproval** and click OK.
4. Select the Implementation tab. In the lower pane, enter the following code:

```
INFER
    ApprovalRule()
    DisapprovalRule()
    backwardchain(->Approval)
END
```

**Note:** The rule methods order in the INFER block is important. Unless the rule order is explicitly overridden by rule priorities, the inference engine considers the rules in the order the rules are posted.

Post DisapprovalRule after ApprovalRule.

5. Select the Properties tab.
6. For the Access Type, select the Public option.
7. Save the changes by clicking "check mark" on the Method Editor.
8. Click the Save icon on the toolbar, and close the Method Editor.

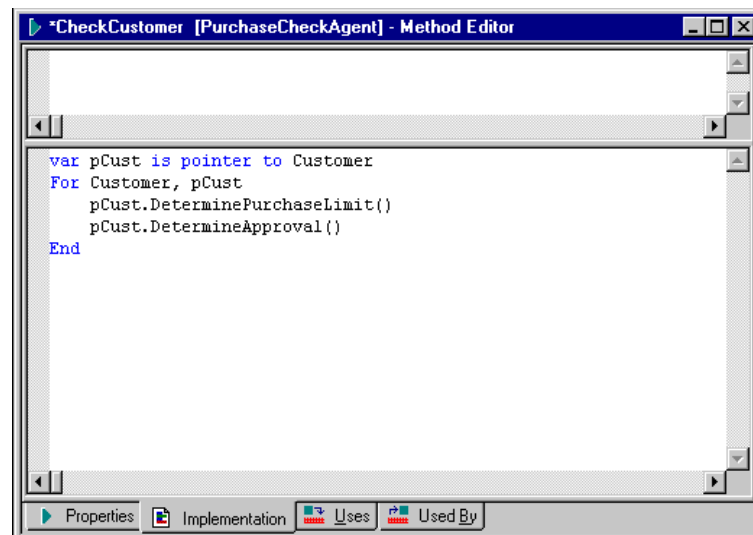
## Call and Test the Rules



To approve or disapprove purchases, you must invoke the new `DetermineApproval()` method from the `CheckCustomer()` method in the `PurchaseCheckAgent` class.

### To call and test the rules

1. Expand the `PurchaseCheckAgent` class and double-click on the `CheckCustomer()` method.
2. The Method Editor appears.
3. After the line `pCust.DeterminePurchaseLimit()`, add the following line of code:

```
pCust.DetermineApproval()
```



4. Save the application by click "check mark" icon on the Method Editor.
5. Click the Debug  button.  
The Aion Debugger dialog appears.
6. Click the Step Into  button several times to verify the new rules.

## Report the Approval/Disapproval Decision

Only one requirement of the system remains. You must report the credit limit and approval decision for each customer. To do this, simply use the Aion `MessageBox()` method to display a message after each customer is processed.

**Note:** Do not use the `MessageBox()` method in a production system. Instead, generate reports. For information about creating reports with CA Aion BRE, see the *CA Aion BRE Product Guide*.

The input arguments of the `MessageBox()` method consist of a message formatted as single string and a title for the message box. These arguments are sufficient to create a message box with an OK button. For more information about the `MessageBox()` method, see the CA Aion BRE online help.

To report the credit limit and approval or disapproval of customer purchases, you must create a method that generates the message, and then invoke the new method.

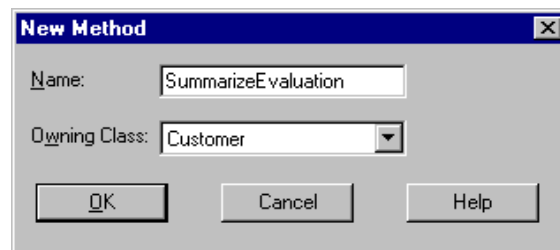
### Generate Message Text

In this procedure, you create a method that returns a string and formats the message.

#### To generate message text

1. Select the Customer query class.
2. Choose Logic, New, Method.

The New Method dialog appears.



3. For the Name field, enter **SummarizeEvaluation** and click OK.
4. Go to the Properties page, set the type to String, and make it Public.
5. Select the Implementation tab.
6. Enter code to create the message.

Here is an example:

```
var Desc is String
Desc = "For Customer: " &Name &CHAR_LF
Desc = Desc &" Marital Status: " &MaritalStatus
      &" Income: $"
      &format(Income) &CHAR_LF
Desc = Desc &" Purchase of $" &format(PurchaseTotal)
      &CHAR_LF
Desc = Desc &" Assigned Limit of $" &format(CreditLimit)
      &CHAR_LF
Desc = Desc &" Approved: " &format(Approval)
return Desc
```

## Display a Message Box

Invoke the MessageBox() method using the following code:

```
MessageBox(SummarizeEvaluation(), "Summary Customer Evaluation")
```

Where should you put this code? One alternative is to call it after the INFER block in DetermineApproval(). But, with future enhancements to the system, you cannot be sure that DetermineApproval() will always be the last method called for evaluating purchases. A better alternative is to make the code the third step in the loop in CheckCustomer().

To invoke the message box from CheckCustomer(), you must answer the following questions. If you need help, look at the completed tutorial in *installation\_directory*\examples\purcheck\purcheck.app.

- What must be true of the SummarizeEvaluation() method for it to be invoked from the PurchaseCheckAgent class?
- How must the previous MessageBox statement be changed so that SummarizeEvaluation() can be invoked from PurchaseCheckAgent?

After you have answered these questions and finished your coding, your application is complete and ready to run by choosing File, Run, Run. If the application does not run correctly, check the application with the debugger.

If you want to decrease the size of the application, go on to the next section. You can choose to stop here, however, because the next procedure is optional.

## Remove an Unused Library (Optional)

CA Aion BRE includes three default libraries in every application: SysLib, DataLib, and WinLib. SysLib is required by every application; DataLib is required for applications involving database connectivity, so it applies to MyPurcheck; and WinLib is required for constructing a Windows GUI interface, which MyPurcheck does not have. Therefore, to reduce the size of the application file, you may want to remove WinLib. This step is optional.

**Note:** You cannot remove a library if the knowledge base has changes that are not saved.

### To remove an unused library

1. Save your application.
2. Choose File, Edit Library Includes.  
The Included Library Editor appears.
3. Select the WinLib library, and click Remove.
4. Click Restore, which restores the application without the library.
5. Click Save.

You have now completed the tour of the CA Aion BRE and you have constructed a CA Aion application using the Aion IDE. While you can run the application in the Aion IDE, this does not mean that your clients can access your work. You have to build the application as an executable, either as a standalone program or as a dynamic link library (DLL) that can be called by another program. CA Aion BRE allows a broad range of options for deploying a CA Aion application into a Windows, UNIX, or mainframe environment.



# Chapter 5: Build and Deploy Applications and Components

---

This chapter provides instructions to build applications for the Windows and UNIX platforms. Deployment on the mainframe requires CA Aion BRE for z/OS and OS/390.

**Note:** For more information about building a CA Aion application for the mainframe environment, see the *CA Aion BRE Mainframe User Guide*.

CA Aion BRE provides facilities for building standalone applications or components from within the Aion IDE on the Windows platform. Additionally, CA Aion BRE for UNIX provides special facilities for building standalone applications and components that will run within a UNIX environment. This chapter includes a tour of building Aion applications on both the Windows and UNIX platforms.

A *component* is a deployable (compiled) software module that communicates with other modules through a well-defined interface. Components are often designed to interoperate as part of a multi-tiered application.

When you build a component from an Aion application, CA Aion BRE creates a dynamic link library (DLL) with the same name as your application file. If you build a standalone application, CA Aion BRE generates an EXE file containing the necessary code for starting the runtime environment. The EXE has the same name as your application file, prefaced by an underscore.

## Build a CA Aion Application under Windows

CA Aion BRE can generate components (DLL files) that support interface layers for a variety of standards, including C, C++, Java, ActiveX, COM (DCOM), and Microsoft Transaction Server.

Building an application, involves these general steps:

- Specify the component interfaces
- Choose the type of component to build
- Build the component

### **More Information:**

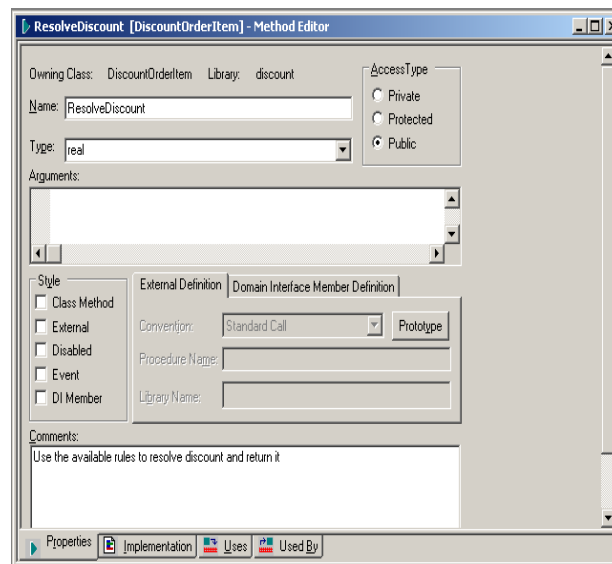
*CA Aion BRE Product Guide*.

## Specify Component Interfaces

Use the Class Editor and the Method Editor to specify the classes and methods to be externally available as interfaces of the component.

### To specify component interfaces

1. In the Class Editor, select the Properties tab.
2. Select the Export check box.
3. In the Method Editor, select the Properties tab.
4. Specify the methods of the classes to be exported by selecting the Public option.



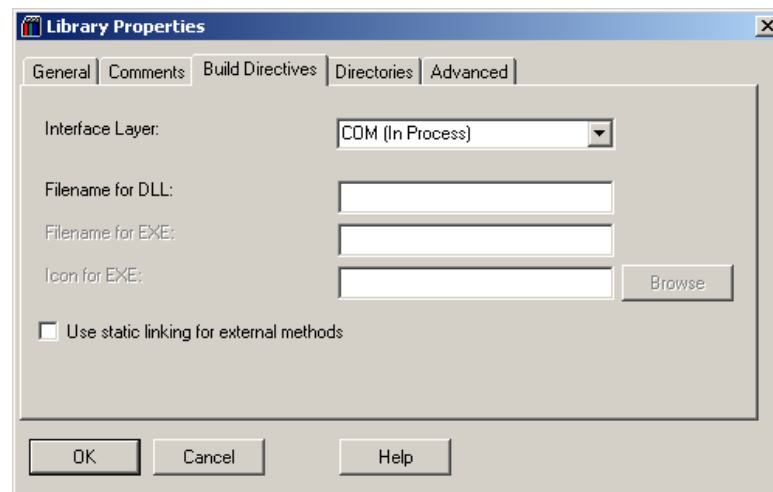


## Choose the Type of Component to Build

### To specify the build properties:

1. In the Project Workspace, select the Libraries tab.
2. Right-click Classes.  
A pop-up menu appears.
3. Choose Properties.

The Library Properties dialog appears.



4. Select the Build Directives tab.
5. From the Interface Layer drop-down menu, choose the component type to generate.

## Build the Component

If you have a compiler installed, you can build the application.

### To build the application, follow these steps:

1. Choose File, Build.
2. To see the build results, examine the Output window (at the bottom of the window).

**Note:** By default, the files required to run your application are in the *appname*\bin subdirectory of your application directory.

## Develop Applications for UNIX Deployment

You must use the Windows-based CA Aion IDE to develop applications for the UNIX platform. Using the Remote Save and Restore features, the CA Aion IDE lets you transfer application files between the Windows platform and any UNIX platform.

More Information:

*CA Aion BRE Product Guide.*

## Develop Applications Overview

### To develop applications for UNIX:

1. Write and debug the application using the Aion IDE.
2. From the Aion IDE, transfer the application to the UNIX platform using the Remote Save feature.
3. To restore and build the application under UNIX, use either the AionBuilder or the command-line utility. These features are discussed in the sections that follow.
4. Review the output to ensure that the compilation was successful.

**Note:** Before transferring an application file developed under Windows to UNIX, ensure that the interface layer specified for the application is compatible with an interface layer supported under UNIX. CA Aion BRE supports C/C++ and Java interface layers under UNIX. Restoring an application with an invalid interface layer causes an error message. It is not possible to fix this problem on the UNIX platform. You must correct the problem under Windows and transfer the application again.

CA Aion BRE for UNIX provides two options for building applications: the AionBuilder and the command-line utilities `respawn` and `reexec`.

## Build Applications with AionBuilder

CA Aion BRE provides the AionBuilder for debugging and compiling applications on UNIX platforms. The AionBuilder offers the options on the File menu of the Aion IDE. You can restore applications, run them interpretively, and, most importantly, build them. The Settings option on the File menu is similar to the Settings option on the File menu of the Aion IDE, giving you access to Directory, Run, and Build settings. The AionBuilder is a Java program that is invoked by running the RunBuilder script.

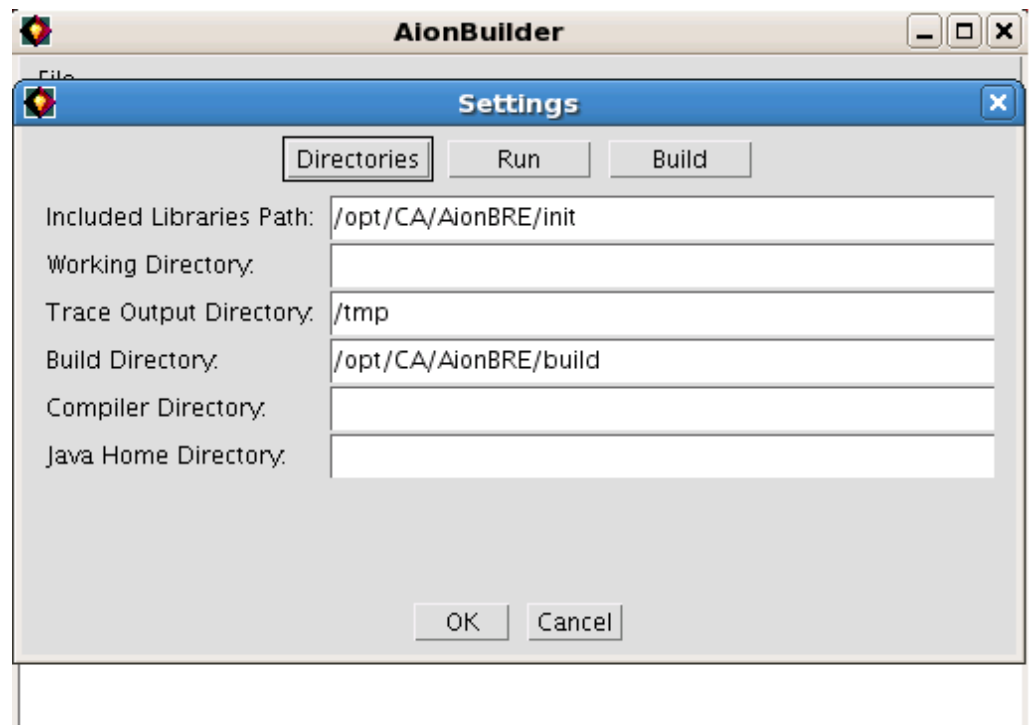
### To launch the AionBuilder

1. From the command line, enter runbuilder.
2. Choose File, Settings.

A dialog appears with functionality similar to the Settings dialog in the Aion IDE.

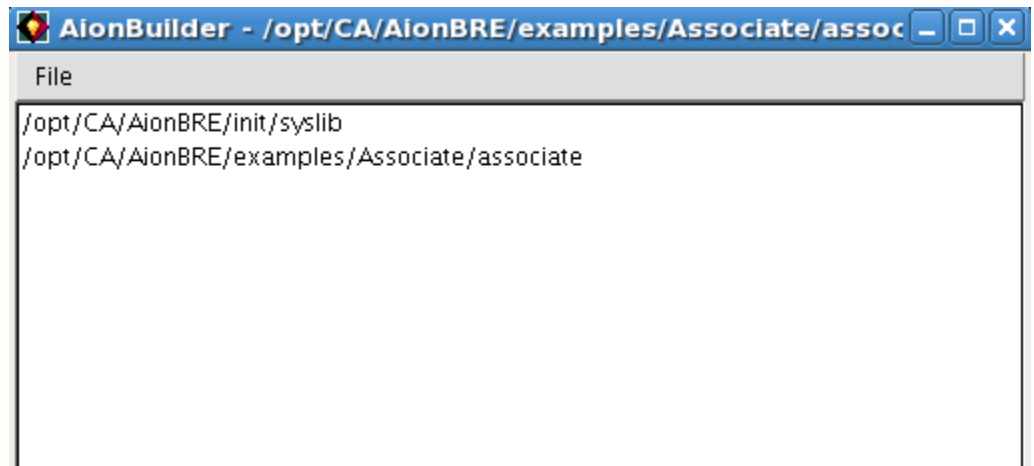
There're 3 buttons "Directories", "Run" and "Build" and let you to set each option

**Note:** For more information about configuring your build, see the *CA Aion BRE Product Guide*.



3. Choose File, Restore from Source (or Open).

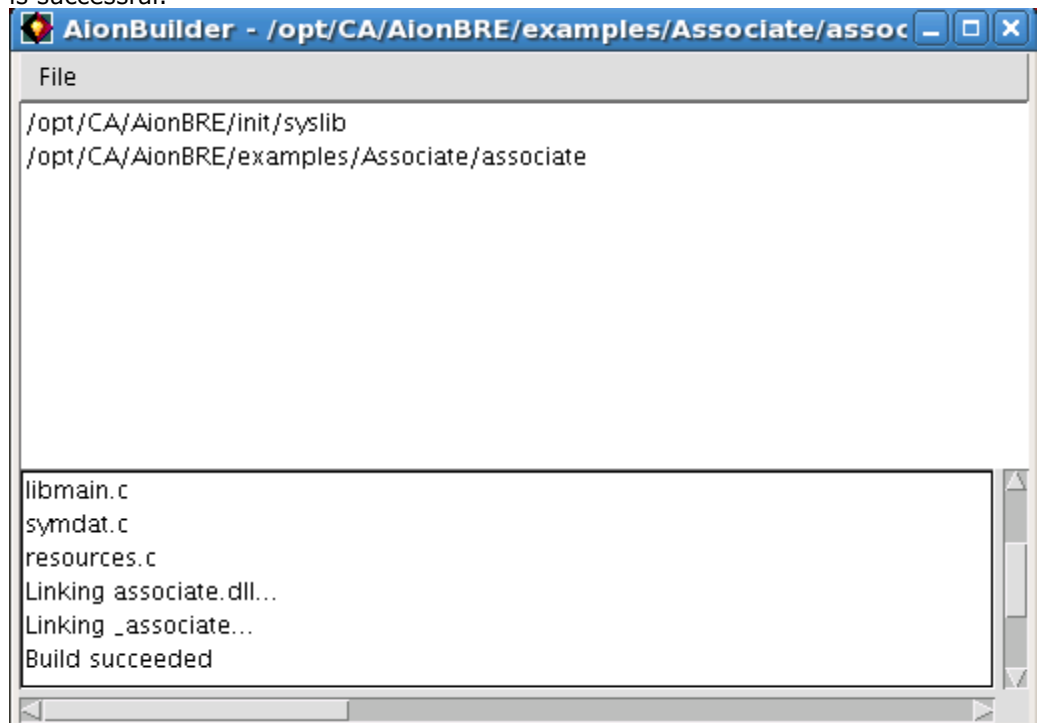
An application opens with the list of included libraries.



**Note:** AionBuilder requires application files to have a lowercase ".app" extension. Do not capitalize the extension to ".APP". When a newer version of an included library is present and the application needs to be restored to reflect those changes.

4. Choose File, Build

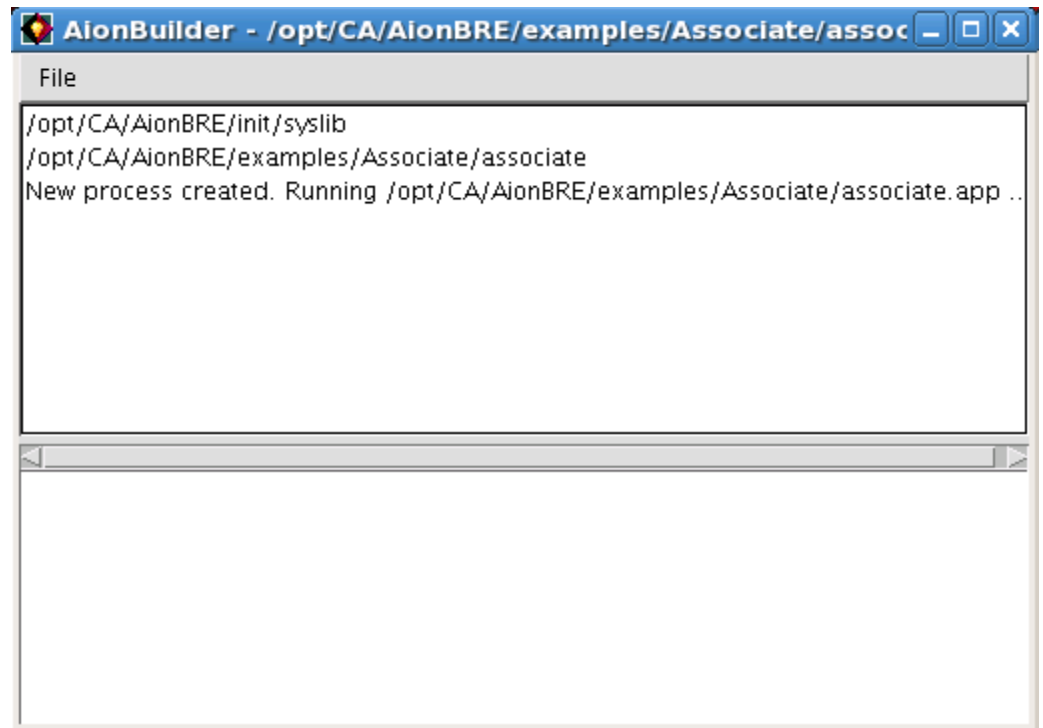
An application is built, the output will be displayed and let you know if the build is successful.



**Note:** The build process is recursive when the Build option is set to Auto. Only the libraries that need to be rebuilt are built. When the Auto option is cleared, the AionBuilder builds only the main application.

5. Choose File, Run.

The application runs.



**Note:** The new process created, the running result is in the terminal, which launches AionBuilder.

## Build Applications with Command Line Utilities

You can compile and run applications on any UNIX platform using utilities invoked from the command line. The respawn utility lets you build applications, and the reexec utility lets you run them interpretively.

### The Respawn Utility

The Respawn utility lets you run build-related functions of the Aion IDE (such as restore, generate code, build, and link) from the UNIX command line. You can build an Aion application by executing the following command:

```
respawn filename.app
```

The Respawn options include:

**Restore**

Restores the Aion application from the .app dataset. The default is FALSE.

**Note:** restore is incompatible with any of the other options, and if it is specified, the file will be restored and any other options ignored.

**Gen**

Generates only C-code. The default is TRUE.

**Comp**

Generates and compiles C-code. The default is TRUE.

**Link**

Links the application objects. The default is TRUE.

**Trace**

Generates trace statements. The default is FALSE.

The executable directory is determined by the value of the #LibExeDir parameter in the application file. This directory is normally set at edit time in the Aion IDE (from the Library Properties, Directories tab), in the AionBuilder, or manually edited in the file. The executable is placed in the directory specified by #LibExeDir, for example, the /exe directory.

**Note:** Respawn requires application files to have a lowercase “.app” extension. Do not capitalize the extension to “.APP.” If you do, you will receive an error message.

## The Reexec Utility

The Reexec Utility runs the application interpretively from the command line:

`reexec filename`

**Note:** Command-line arguments used within the Aion application can also be added to the end of the reexec statement.

## Compilation Hints and Suggestions

This section provides hints and suggestions for ensuring successful compilation of Aion applications under UNIX.

### **File Transfer**

Make sure that filenames under UNIX match the name of the Aion application exactly as stated in the second line of the .app file. When using FTP, make sure that the name of the file preserves the case (mixed case mode) coming from the Windows platform.

### **Compiling Wrapper Programs**

For C++ programs that use an Aion component via a C/C++ wrapper, compile the C/C++ wrapper program using the -DUNIX compile switch. This parameter must be capitalized.

### **Cross-platform Data Access**

When using IOLIB for cross-platform applications, remember that case is important for filenames (File and DirectoryEntry classes).





# Appendix A: The Aion BRE Inference Engine and Rules

---

This chapter describes the CA Aion BRE inference engine and rules.

The Aion BRE *inference engine* is a set of algorithms for determining the order in which a set of non-procedural, declarative statements are to be executed. The inference engine processes rules to infer new knowledge from knowledge that is already known.

Rules do not need to be entered or stored in any particular order. If an application includes all the rules and other knowledge needed to solve a problem, and if the rules are written correctly, the inference engine will execute the appropriate rules when it needs them. If necessary, rules can be ordered so that more important rules are considered first.

## Basic Inferencing Techniques

The two main rule-based inferencing techniques are forward chaining and backward chaining. The basic form of a rule in CA Aion BRE is an IF-THEN statement that can be executed either forward or backward. This appendix presents the following information:

- Structure of an IF-THEN statement
- Explanation of forward chaining and backward chaining
- Types of rules and their syntax

### **More Information:**

*CA Aion BRE Rules Guide.*

## Rule Structure

A basic type of rule in CA Aion BRE is the IFRULE, which is an IF-THEN statement:

*IF condition THEN action*

- The IF clause, or condition (sometimes also called premise), is used to examine attribute values in one or more instances or classes.
- The THEN clause, or action, invokes methods or otherwise causes data in the instances to change.

**Note:** The IF-THEN examples do not reflect the actual rule syntax in CA Aion BRE. The examples are simplified statements that make it easier to illustrate forward and backward chaining.

### Example

The following is an example of an IF-THEN statement:

**IF:**

Client has income > \$40,000 AND

Client is < 50 years old AND

Client is willing to take a risk,

**THEN:**

Add 500 shares of Phelye, Buy, & Knife stock to Client's retirement investment portfolio.

Forward chaining and backward chaining rules in CA Aion BRE have the same general structure, with IF conditions and THEN actions.

## Forward Chaining

Inferencing using forward chaining has the following characteristics:

- It is data-driven.
- The inference engine enters rules through the IF clause (condition).
- The inference engine continues until it infers all possible knowledge.
- Forward chaining is typically used for scheduling and design (configuration) problems. Also, with WHEN and WHENMATCH rules (demons), which are shown later in this appendix, forward chaining is used in exception handling and applications requiring real-time monitoring.

## Example

The following simple example shows a typical sequence of forward chaining events:

**IF**

A driver is assigned AND

A forklift operator is assigned AND

A foreman is assigned

**THEN**

All workers have been assigned to the work shift.

**IF**

All workers have been assigned to the work shift AND

The day is Saturday AND

Funds have been allocated for overtime

**THEN**

Work can begin.

Following is an explanation:

1. The first rule executes when its IF clause (condition) becomes true.
2. The data from the THEN clause (action) makes part of the condition of the second rule true.
3. If the other parts of the condition are also true, the second rule fires.

## Backward Chaining

Inferencing using backward chaining has the following characteristics:

- It is goal-directed.
- The inference engine enters rules through the THEN clause (action).
- Subgoals are automatically set and resolved (if possible). Resolving a goal means finding or deriving a value for the goal, so that it evaluates to something other than unknown.
- Backward chaining is typically used for diagnostic and classification purposes.
- In backward chaining, the inference engine starts with a processing goal and works backward through the rules to determine values that can resolve the goal.

## Example

The following simple example shows a typical sequence of backward chaining events. It is the same as the example for forward chaining, but it goes from bottom to top.

**IF**

A driver is assigned AND

A forklift operator is assigned AND

A foreman is assigned,

**THEN**

All workers have been assigned to the work shift.

**IF**

All workers have been assigned to the work shift AND

The day is Saturday AND

Funds have been allocated for overtime,

**THEN**

Work can begin.

Here is an explanation:

1. A goal (can work begin?) is posed to the inference engine. If it cannot find a resolution already in the knowledge base, it looks for a rule with a matching action.
2. If the condition of a rule that the inference engine is testing is true, the rule applies (it is "fired"), and the action (THEN) is executed thereby establishing a value for the goal.
3. If the condition of a rule that the inference engine is testing also contains unknown attributes, these attributes are posed to the inference engine as new goals (subgoals), and the inference engine will then pursue these subgoals in order to determine whether the rule applies.

## Aion BRE Rule Types and Syntax

In CA Aion BRE , rules are IF-THEN statements and variants of those, such as IFMATCH and WHEN rules. The following table shows the types of rules and their syntax:

Type of Rule	Example	Description
IFRULE	IFRULE theValue =	IFRULEs set the values on instance

Type of Rule	Example	Description
	<pre> "High" and (theRisk = "High" or theRisk = Med") THEN theConcern = :High" END </pre>	<p>attributes or class attributes according to the instance or class to which they are bound when they are posted.</p> <p>IFRULEs can be executed with either forward or backward chaining.</p>
IFMATCH	<pre> IFMATCH aTask, aResource WHERE aTask.Requirement = aResource.Ability THEN aTask.Assign(aResourc e) END </pre>	<p>IFMATCH rules are used to manage sets of instances. An example is joining instances based on criteria for matching instances of two different classes (the WHERE clause). The action of the rule is invoked for the matching instances. IFMATCH rules are also known as <i>pattern matching rules</i>.</p> <p>IFMATCH rules can be executed only with forward chaining.</p>
WHEN	<pre> WHEN theValue = "High" THEN theConcern = "High" END </pre>	<p>WHEN rules are called demons. They monitor the state of the instance or class to which they are bound when they are posted. Demons fire whenever their premise becomes true, no matter what else is going on in the system.</p> <p>Demons are executed with forward chaining.</p>
WHENMATCH	<pre> WHENMATCH aTask WHERE Available = True THEN aTask.ScheduleTask() END </pre>	<p>WHENMATCH demons monitor the states of instances in a set by using pattern matching. In the example, when the Available attribute of an instance of Task is set to True, the ScheduleTask () method is executed for that instance.</p> <p>WHENMATCH demons are executed with forward chaining.</p>

As stated in the previous table, rules (especially IFRULEs and WHEN demons) are bound to the instance to which they are posted. Posting is the process by which the programmer tells the inference engine which rules to consider at a specific time:


- In CA Aion BRE , rules are organized into methods known as *rule methods*. A rule method looks like a regular method except that it contains declarative code (rules) in the syntax shown in the previous table rather than in procedural code.
- When a rule method is executed, the rules in that method are “posted” (although no inferencing is invoked yet).
- When posted, all rules are bound to the instance or class that “owns” the rule method that contains the rules. That is, each rule knows the meaning of *current* (in C++, *this*), or *currentclass*. This binding is particularly important in the case of IFRULEs and WHEN demons; all attributes in those rules automatically refer to the current instance or class of those rules. Thus, the same rule, when posted from different instances, sets the value of the same attribute in different individuals.
- Rules bound to objects constitutes one of the critical contributions of CA Aion BRE to object-oriented programming: the notion of rule-based behavior of individuals. Individuals encapsulate the rules of their behavior.
- In CA Aion BRE, the ability of the programmer to choose which rules (behavior) of an instance or class to post is known as *dynamic inferencing*. Dynamic inferencing promotes inferencing efficiency because the programmer can control whether to post rules from one method and not another. For example, rules for evaluating medical reports for males may be kept in one rule method and rules for evaluating medical reports for females may be kept in another. The programmer can write code (as simple IF statement) to test whether the current patient is a male or a female and post only the rules of the relevant rule method.

# Appendix B: The Aion BRE Debugger

This chapter will help to find and eliminate any CA Aion application problems by using the *Aion BRE* Debugger tool. When you use debug mode, the application runs as usual: data is loaded, windows appear, and values are calculated. However, you can monitor the flow of execution, method-by-method or line-by-line. You also can see the values of variables.

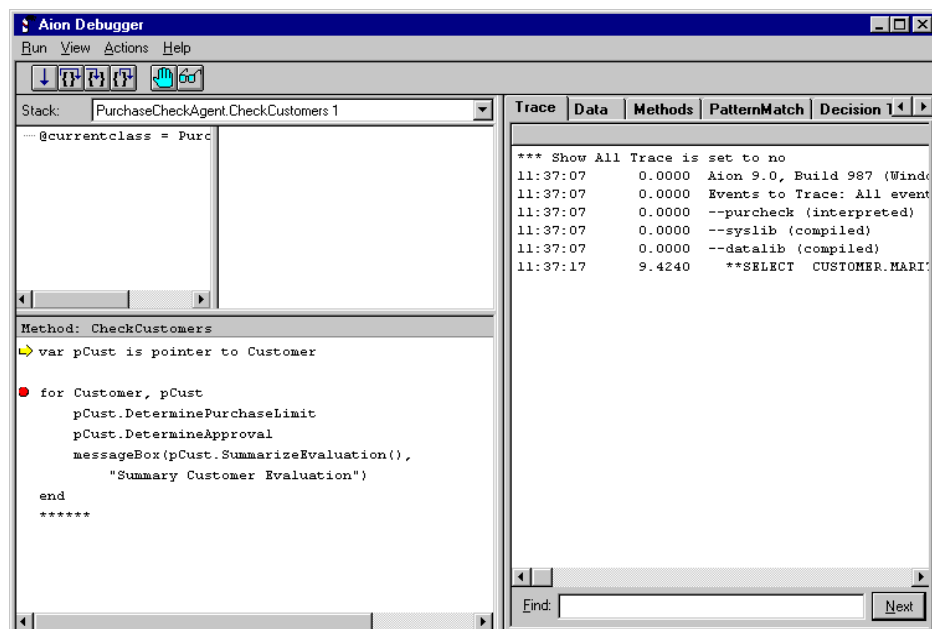
For more information about the *Aion BRE* Debugger, see the *CA Aion BRE Product Guide*.

## The Debugger Button

The Debugger  button is located on the Aion IDE toolbar.

To open the Debugger window, click the Debugger button.

## The Debugger Window



The features of the Debugger Window include:

### **Stack List Box**

This listbox displays the Call Stack, which is a list of all method calls that you have not returned (in other words, suspended methods). By default, the currently executing method is displayed as class, method. The code for this method is shown in the Method Body Pane. During inferencing, the Stack List Box displays inferencing information instead of class, method.

### **Arguments Pane**

This pane displays the values for all input, output, and local variables in the currently executing method. To expand an attribute, click the plus icon. To display or modify the current value, double-click it.

### **Watched Attribute Pane**

This pane displays the values of any attributes and instances for which watchpoints or data breakpoints have been set. For information about setting watchpoints, see Set Breakpoints and Watchpoints in the appendix "The Aion BRE Debugger." For information about setting data breakpoints, see the *CA Aion BRE* online help.

### **Debugger Tab Pages**

These pages show various types of information about the application being debugged.

### **Methods**

Lists the current application and all included libraries that are being run interpretively. It shows the classes in those libraries and the methods within each class. Here, you can set code breakpoints and see the list of all breakpoints.

### **PatternMatch**

Shows the internal processing of patternmatching rules (IFMATCH and WHENMATCH). Values appear during forward chaining only.

### **Decision Table**

Displays a graphical view of an active decision table. (The decision rule appears in the Method Body Pane).

### **Method Body Pane**

This pane contains code of the current method. (The method name is listed in the Stack List Box). The yellow arrow indicates the next statement to be executed; a red flag signifies a code breakpoint.

### **Instance Counter**


This area on the Status Bar shows the current number of dynamic instances in the application. Static instances are not included.



### Find in Trace Text

This box appears on the Trace page only. This Find feature lets you search for a word, letter, or number on the Trace page.

## Open the Debugger Window

To open the Debugger window, click the Debugger  button. The Debugger window appears.

## The Debugger Toolbar

The toolbar buttons on the Aion BRE Debugger let you step through methods and set code breakpoints and data watchpoints.

### Step through Methods

The toolbar buttons let you proceed through the debugging process:



#### **Continue Execution**

Proceeds to the next breakpoint in the application (same as F5).



#### **Step Over**

Executes the next statement in the method but does not display internal logic (same as F10).



#### **Step Into**

Executes the next statement in the method and displays internal logic (same as F11).



#### **Step Out**

Jumps out of the current method and returns to the calling method (same as Shift F11).

## Set Breakpoints and Watchpoints

The other buttons on the toolbar let you set breakpoints on methods and watchpoints on attributes:



### **Toggle Breakpoint**

A breakpoint suspends the execution of a method. You can put a breakpoint on any line of code in a method.

#### **To add a breakpoint**

1. Select the Methods tab in the Debugger, and expand the library and class that contain the method in which you want to put a breakpoint.
2. Select the method. Its code appears in the lower right pane of the Method page.
3. Click the line to receive the breakpoint, and then click the Toggle Breakpoint button (same as F9).


**Note:** You can toggle a breakpoint on and off by double-clicking the line or by clicking Toggle Breakpoint.




### **Toggle Watchpoint**

A watchpoint lets you monitor the values for an attribute. All attributes with watchpoints appear in the Watched Attribute pane at the top middle of the Debugger.

#### **To add a Watchpoint**

1. Click the Data tab, and expand the library that contains the attribute you want to watch.
2. To place a watchpoint on a class attribute (  ), click the name of the class. Its class attributes appear in the lower right pane.

To place a watchpoint on an instance attribute (  ), expand a class to display its instances. Click the instance whose attribute you want to watch.

The instance attributes appear in the lower right pane.

3. Do *one* of the following:
  - Select the name of the attribute you want to watch, and then click Toggle Watchpoint on the toolbar.
  - Right-click the name of the attribute.  
From the pop-up menu, choose Toggle Watchpoint.
  - Choose Actions, Attribute, Toggle Watchpoint.

# Index

---

## A

### actions

- change information for • 18
- create • 18
- delete • 18
- display • 18

Aion Debugger dialog • 60, 66

Application Assistant dialog • 41

APPWindow class, expand the • 23

assign values to attributes • 24

### assistants

- use of • 15

### attribute

- assign a value to an • 24

attributes, create • 49

## B

### backward chaining

- explanation of • 81
- use of • 83

BackwardChain command • 21

business client user role • 10

## C

### CA Aion BRE

- architecture • 9
- deploy as a batch program • 9
- description of • 9
- graphical user interface (GUI) tools • 9

calculate credit limit, create method to • 50

change information for actions • 18

CheckCustomer() method • 60

CheckForSpecialDiscount() method • 23

### commands

- BackwardChain • 21
- ForwardChain • 21, 22

### conditions

- create • 18
- delete • 18

Conditions/Actions tab • 18

Confirmation dialog • 31

Connection Properties dialog • 31, 42

### create

- and maintain static rules • 13

attributes to store credit limit and approval • 49

conditions and actions for decision tables • 18

decision table • 52

knowledge base • 41

method to calculate credit limit • 50

Create Accessors dialog • 50

Create New Data Source dialog • 29

Customer.Load() method • 60

## D

### data sources

- specify • 29
- test • 31

Data Test dialog • 32

Database Assistant dialog • 31, 41

### decision table

- call a • 58
- create a • 52
- execute • 60
- post a • 57
- test a • 60

### Decision Table Editor

- open the • 18
- use of • 18

Decision Table Editor dialog • 52

### decision tables

- benefits of • 21
- create • 18
- definition of • 18
- delete conditions and actions for • 18
- sample • 18

declarative code (rules) • 21

demons • 82, 84

DetermineAccountStatus() method • 23

### develop

- applications for UNIX deployment • 74
- business intelligence application • 10
- knowledge base • 40

### dialogs

- Aion Debugger • 60, 66
- Application Assistant • 41
- Confirmation • 31
- Connection Properties • 31, 42
- Create Accessors • 50

---

- Create New Data Source • 29
- Data Test • 32
- Database Assistant • 31, 41
- Decision Table Editor • 52
- Find • 22
- Library Properties • 73
- New Application • 41
- New Attribute • 49
- New Decision Table • 52
- New Method • 50, 58
- New Query • 43
- New Rule Method • 62
- ODBC Data Source Administrator • 29
- ODBC Microsoft Access Setup • 29
- Properties • 45
- Select Database • 29
- Settings • 75
- DiscountOrderItem class, expand the • 25
- domain interface
  - members
- definition of • 18
- dynamic
  - inferencing, definition of • 84
  - rules
- definition of • 13
- use of • 10

## E

- END procedural methods • 21
- execute the decision table • 60

## F

- Find dialog • 22
- find feature, use of • 22
- forward
  - chaining
- example of • 83
- execute demons with • 84
- execute MATCH rules with • 84
- explanation of • 81
- use of • 82
  - mode, use of • 24
- ForwardChain command • 21, 22

## I

- IF
  - clause or condition • 82
  - condition, example of • 82
  - conditions • 18

- IFMATCH rule • 15
- IFRULEs, definition of • 14
- IF-THEN statements
  - description of • 14
  - example of • 82
  - execute • 81
  - list of • 84
  - structure of • 82
- INFER procedural methods • 21
- inference
  - blocks
- definition of • 21
- methods that contain • 23
- multiple • 21
- view • 22
  - engine
- definition of • 81
- enter rules • 82, 83
- pose goals and attributes to the • 84
- processing rules • 13
  - methods, definition of • 21
- input argument, location of an • 18
- invoke inferencing • 21
- IT developer role • 10

## K

- knowledge base, develop a • 40

## L

- libraries page • 22
- Library Properties dialog • 73
- list of
  - IF-THEN statements • 84
  - operating systems • 9
  - rule types and syntax • 84
  - rules • 21

## M

- Method Editor
  - open the • 14
- methods
  - CheckCustomer() • 60
  - CheckForSpecialDiscount() • 23
  - Customer.Load() • 60
  - DetermineAccountStatus() • 23
  - ResolveDiscount() • 23
  - RulesAboutTotalDiscount() • 25
  - Season() • 18
  - SetExplanation() • 22

---

SetSeasonalDiscount() • 18  
WhenpbDiscountsChosen() • 23

## N

New Application dialog • 41  
New Attribute dialog • 49  
New Decision Table dialog • 52  
new goals, pose • 84  
New Method dialog • 50, 58  
New Query dialog • 43  
New Rule Method dialog • 62

## O

ODBC Data Source Administrator dialog • 29  
ODBC data source, defining an • 29  
ODBC Microsoft Access Setup dialog • 29  
open  
    Decision Table Editor • 18  
    the Method Editor • 14  
    the Rule Analyzer • 25  
operating systems, list of • 9

## P

pattern matching rules See also IFMATCH rules  
    • 84  
premise See also IF clause or condition • 82  
procedural code (operations) • 21  
Properties dialog • 45

## R

remove a library from a knowledge base • 69  
ResolveDiscount() method • 23  
rule  
    analyzer, the • 24  
    editor  
access the • 18  
close the • 15  
display the • 15  
modify IFRULEs with the • 14  
open the • 15  
sample of the • 25  
use of • 15  
    methods  
and procedural methods, differences between • 21  
definition of • 14, 84  
    programming techniques • 13  
    types and syntax, list of • 84  
rules

Aion BRE inference engine • 81  
and rule methods • 14  
and their syntax, types of • 81  
apply (fired) • 84  
IFMATCH • 15, 84  
IFRULE • 14, 84  
IFRULEs • 84  
meaning of current or currentclass • 84  
posting • 21  
WHEN • 82, 84  
WHENMATCH • 82, 84

Rules tab, access the • 15  
RulesAboutPreferredCustomer rule method • 15  
RulesAboutTotalDiscount() method • 25  
runtime algorithms • 21

## S

Season() method • 18  
Select Database dialog • 29  
SetExplanation() method • 22  
SetSeasonalDiscount() method • 18  
Settings dialog • 75  
static rules  
    create and maintain • 13  
    definition of • 13  
subgoals, pose • 84  
switch between forward and backward mode • 25

## T

THEN  
    action, example of • 82  
    actions • 18  
    clause or action • 82, 83

## U

use  
    backward chaining • 83  
    databases with CA Aion BRE • 29  
    forward chaining • 82  
    FTP • 79  
    pattern matching • 84  
    the Aion BRE Debugger tool • 87  
    the AionBuilder • 75  
    the -DUNIX compile switch • 79  
    the Method Editor • 64  
    the ODBC Data Source Administrator • 29  
    the Remote Save and Restore features • 74  
use the forward mode • 24

---

## V

view

- inference blocks • 22

## W

WHEN rules • 82

WHENMATCH rules (demons) • 82

WhenpbDiscountsChosen() method • 23