

CA Aion[®] Business Rules Expert

Product Guide

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 © 2009 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.

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

Chapter 1: Introduction	19
Rules and Inferencing	19
Full Object Orientation	20
Component-Based Development	21
Multiple Application Architectures	21
Graphical User Interface Builder	22
Visual Editors and the Method Editor	22
Aion BRE Language	22
Database Support	23
Rapid Application Development	23
Chapter 2: Installation and Setup and Uninstall	25
Installation Prerequisites	25
Install on Microsoft Windows	26
Custom Installation	29
Uninstall on Microsoft Windows	30
Install on Linux/UNIX server	31
Console-Mode Installation	31
Graphical-Mode Installation	32
Uninstall Linux/UNIX server	39
Silent Installation	40
Chapter 3: Overview of CA Aion BRE Objects	43
Applications	43
Libraries Are Functional Units	44
Included Libraries	44
Stand-Alone Aion BRE Applications	45
Benefits of Included Libraries	45
Library Boundaries	46
Object Orientation	47
Basic Object-Oriented Terms	47
Apply the Object-Oriented Paradigm	48
Inheritance	49
Polymorphism	50
Associations	51
Class Containment	52

Attached Objects	57
Constraints	59
Access Types	61
Comparing Terms	62
Anatomy of a Class	63
Attributes	63
Methods	67
Constants	69
Dynamic versus Static Instances	70
Static Instances	71
Dynamic Instances	71
Implement Interfaces	72
Develop Interfaces	72
Associate Interfaces to Classes	74
Associate Interfaces to Class Instances and Generic Methods	75
Implicit Typecasting	76
Associated SysLib Methods	78
Inference Considerations	78

Chapter 4: How You Create and Edit Applications **79**

Create Applications	79
.APP and .BIN Files	80
Set Mainframe Line Length	80
Open Applications	80
Save Applications	81
Read-Only Applications	81
Back Up Applications	81
Restore Applications	82
Restore Closed Applications	83
Restore Open Applications	83
Develop Applications for Non-Windows Platforms	84
The Command Line	88
Customize the Development Environment	89
Directories Tab Page	90
View Applications	91
Manipulate Windows	91
Object Icons	92
Fonts	93
The Output Window	93
Output Window Tab Pages	94
The Project Workspace	96
View Inheritance and Ownership Information	96

Project Workspace Tab Pages	96
The Explorer	99
Set Explorer/Workspace Options	100
The Rule Analyzer	102
Editors	102
Standard Tab Pages	103
Standard Procedures	103
Association Editor	104
Attribute Editor	105
Class Editor	106
Decision Table Editor	106
Instance Editor	107
Menu Editor	107
Method Editor	107
Rule Editor	108
Query Editor	109
Stored Procedure Editor	110
Tool Editor	111
Window Editor	111
Create a Constrained Attribute	112
Specify a Constraint	113
Attribute Declarations Using Constrained Data Types	114
Operation of Constraints	115
Copy and Paste Objects	117
Delete Objects	118
Edit Toolbars	118
Supplied Toolbars	119
Customize Toolbars	122
Work with Included Libraries	122
Include and Remove Libraries	123
Work with Source Control	126
Set Source Control Safeguards	127
Source Control Menu Options	128
Enable Concurrent Development	129
Change Management	130
Change Management Functions	130
Produce Reports for an Application	133
Printed Report Contents	134
CA Aion BRE Documentation	134
Search for Objects by Name	136
Search for Objects by Multiple Criteria	136
Replace Text	136

Search Across Applications	137
----------------------------------	-----

Chapter 5: Create a Graphical User Interface **139**

How You Create a GUI	139
Step 1-Define data requirements.....	139
Step 2-Construct the application's main frame window.....	140
Step 3-Create dialog boxes.....	140
Step 4-Add logic.....	140
Work with Object Properties	141
Open and Use Properties Dialog Boxes	141
Work with Windows and Dialog Boxes	146
Subclass Supplied Classes	146
Multiple Document Interface	148
Create Windows and Dialog Boxes	149
Edit Windows and Dialog Boxes	149
Standard Window and Dialog Box Properties	150
Add Controls to Windows.....	151
Controls Supplied by CA Aion BRE.....	152
Create Controls	154
Edit Controls	155
Order Keyboard Focus.....	156
Work with Control Groups	156
Radio Buttons	156
Tab Controls	158
Add ActiveX Controls to Your GUI	159
Insert OLE Objects into an Application.....	160
Splitter Windows.....	160
Control Properties.....	161
Add Menus to Windows	163
Create Menu Titles	164
Add Menu Items to Menu Titles	165
Attach Menu Titles to Windows	166
Pop-Up Menus	166
Menu Properties	167
Menu Conventions	171
Add Toolbars to Windows	171
Create Toolbars.....	171
Add Tool Items to Toolbars	173
Attach Toolbars to Windows	173
Toolbar Properties	173
Toolbar Conventions	175
Add Graphics to Windows	175

Layered Graphics	176
The Order Controls	176
Create Resources	177
Graphics Properties	177
Implement Logic to Run Your GUI	178
Window and Dialog Box Logic	179
Control Logic	182
Menu Logic	183
Toolbar Logic	185
Graphic Logic	185
Conclusion	186

Chapter 6: Write Logic **187**

Write Application Logic	187
Sample Applications	188
About Methods	188
Library Methods	189
Event-Triggered Methods	189
External Methods	189
Method Editor	190
Open an Existing Method in the Editor	190
Create a New Method	190
Specify a Method's Properties	191
Specify the Method's Implementation	195
Parse and Save a Method	196
How You Program Aion BRE	197
Arguments in Method Calls	197
Attribute Data Types	198
Local Variables	200
Return Values	201
Call an Instance Methods	202
Call a Class Methods	203
Associations	203
Attribute and Class Pointers	205
Specialize a Method	206
Unspecialize a Specialized Method	207
Write Logic for Windows and Dialogs	207
Create and Open the Application Window	207
Create Dialogs at Runtime	208
The DialogBox Class	208
Set Initial Values for Controls	209
Use Dialogs to Get User Input	210

Report Status Using a Modeless Dialog	211
Process Data	211
Iterate Instances of a Query	212
Use Markers to Control Data Selection.....	212
Define Other Objects	213
Attribute Editor	213
Accessor Methods	214
Instance Values Dialog	214
Use the Language Paster	215

Chapter 7: Access Data **217**

Data and Aion BRE Classes	217
Data Manipulation	218
Update the Database.....	218
Basic Steps in Working with Data	218
Define a Database Connection	222
Define a Query	222
Use Inheritance to Reuse Queries	223
Concurrency Control	224
Create a Query	224
Query Editor.....	224
Field Attributes	226
Use Markers with Query Classes	227
Change the Properties of a Query	228
Dynamic versus Static SQL	228
Build Queries Using Static SQL.....	230
Write SQL Statements	231
SQL Paster Utility	232
Define a Stored Procedure	232
Queries and Stored Procedures	233
Create Stored Procedures	233
The Stored Procedure Editor	233
Add Markers to Stored Procedures	235
Data Test Facility	236
Select Statement Field	236
Markers List	237
Result Set Field	237
Load Data from a Database	237
Load Data	238
Cursor Management	238
Manual Load Mode	239
WhenFetched()	241

Save Modifications to the Database	243
Data-Update Mechanisms	244
Update Data with Automatic and Manual Commit Modes	244
Automatic Commit Mode	245
Manual Commit Mode	245
WhenUpdated()	246
Database Errors	248
Define Records and Serialize Data	249
Construct Records	249
Record Elements	251
Serialize Data	253
MQLib to Access MQSeries	253
Code the Queue Manager	254
MQLib Data Objects	255

Chapter 8: Process XML 257

SAXLib - Read XML Documents	258
How SAXLib Functions	259
Attributes Class	261
Process Exceptions	261
Use SAXLib	262
Use the SAX API	263
DOMLib - Read and Write XML Documents	265
Initialization	267
Process the DOM Tree	268
XML Maintenance Using DOMLib	269
Add Elements to an XML Document	269
Delete Elements from an XML Document	270
Create an XML Document	271
Handle Character Data as Element Values	273
Generate Applications Based on XML Schemas	273
General Approach	274
Details	275
Use the XsdConverter	277
Process the XML Document with the Generated Application	278
Read an XML Document	279
Write an XML Document	279
Update an XML Document	280
Automatic Unmarshalling and Marshalling	281
Load() Method	281
Dump() Method	282
The Purchase Order Example	282

Chapter 9: Domain Interfaces and Dynamic Rules	285
Domain Interfaces	286
Role of the Domain Interface in System Development	288
Create Domain Interface Members	290
Dynamic Rules	291
Useages for Dynamic Rules	292
Dynamic Rules Task Flow	293
Support for Dynamic Rules: Aion BRE-Supplied Libraries	294
External Rules: Use Dynamic Rules or Generate Static Rules with COBSLib?	295
Accommodation of Rules for Any Format	296
No Performance Degradation	296
Immediate Use of External Rules	296
Runtime Loading of Rules	297
Styles of Rules	297
Chapter 10: Use the Rule Manager Wizard	299
Process Overview	299
Invoke the Rule Manager Wizard	300
Chapter 11: Dynamic Rule Management	303
Rule Repository Functionality	304
Set Up the Rule Repository	304
The Business Rule Management Process	305
Establish User Access Permissions	306
Dynamic Rule Repository Functionality	307
Business Rule Maintenance Scenarios	312
Chapter 12: Aion BRE Reports	315
About Aion BRE Reports	315
Aion--IOLib and IOWLib	316
Aion--How you Create a Report	316
Aion--IOLib Classes	317
Canvas	317
Artist	318
Fine Artist	318
HTMLArtist	318
IOWLib Classes	319
WindowArtist	319
PrinterArtist	319
Work with a Report Canvas	319

Create the Canvas Instance	320
Start the Report Page	320
Define the Overall Appearance	320
Specify Font Attributes	322
Write Text and Images to the Canvas	322
Add Blank Lines	323
Specify Tables	324
End the Report	324
How You Use the Artists	325
Create the Artist Instance	325
Specify the Output Device	325
Render a Canvas	327
Sample Application	327
The Sample Canvas	328
Output Options	329
Output the Report to a Window	330
Output the Report to a Text File	331
Output the Report to an HTML File	332
Send the Report Output to a Printer	333

Chapter 13: Generate and Use C and C++ Components **335**

Build an Aion BRE Component with an Interface Layer	335
Invoke Aion BRE Methods from C/C++ Clients	336
Use Exported Aion BRE Methods in a C Program	337
Use Exported Aion BRE Methods in a C++ Program	339
Invoke C Functions from Aion BRE	339
How to Create an External Method in Aion BRE	340
Call an External Method (Runtime)	341
Data Type Mappings	341
Input Arguments	342
Output Arguments	342
Return Values	342
Mapping Between Aion BRE and C Data Types	342
Aion BRE Strings in C and C++	343
NULL Values	349

Chapter 14: Generate and Use Managed C++ Components **351**

The Managed C++ Interface Layer	351
Managed Code	351
Structure of the Managed C++ Interface Layer	353
Set up the Environment	355

Create an Aion BRE Component with the Managed C++ Interface	355
Code an Exported Class	355
Build the Aion BRE Application	356
Write and Compile the .NET Client	356
Deploy a Managed C++ Component	357
Application Programming for .NET: The Basics	357
Support for Output Parameters	358
Support for Complex Data Types	360
Object Management Under .NET	360
Data Type Conversion	362

Chapter 15: Generate and Use Java Components **363**

The Java Interface Layer	363
Elements in the Java Interface Layer	364
Set Up the Environment	366
Create an Aion BRE component Using the Java Interface	366
Code an Exported Class	367
Build the Aion BRE Application	367
Write and Compile the Java Client	368
Test the Java Interface	369
The Basics of Java Application Programming	369
Java Objects	370
Data Conversions and Exception Handling	371
Java Object Management	373
Supports Backward Chaining	374
Aion BRE Deployment on the Web	376
Roles and Responsibilities	376
Servlet Technology	377
Java Servlet Programming Considerations	380
Thread Management	382
General Definitions	383
Support for Concurrency and Session Safety	384
Resource Load Issues	385
Additional Information	385

Chapter 16: Generate and Use COM Components **387**

Automation	388
ActiveX	388
DCOM	388
MTS	389
COM+	389

Aion BRE and COM.....	389
Aion BRE and COM+	389
Object Generation Overview	390
Set Up the Environment	390
COM Object Generation	390
ActiveX Object Generation	391
MTS Object Generation	391
COM+ Object Generation	391
MVS COM Object Generation	391
Generate COM or ActiveX Objects	392
Data Type Support in Automation Servers	392
Register the COM Object	393
Configure DCOM	393
Test the DCOM Configuration	393
COM+ Application Configuration	394
Use Aion as an Automation Client or Server	394
Include COM Objects in Aion Applications	394
Data Type Support in Automation Clients	395
AutoLib Example	395
Start() Method in client.app	396
Implement Callbacks Between COM Servers	396
COM Interface Server Side Example	397
COM+ Services: Server Side	398

Chapter 17: Deploy Aion BRE Components as Web Services **399**

Program Aion BRE Components as Web Services	399
Use Complex Data Types	400
Program Standards for Web Services	400
Do I Need To Install Apache Axis?	401
Validate the Apache Axis Setup	402
Generate an Aion BRE Component as a Web Service	403
Deploy Aion BRE Web Service Components on Microsoft Windows	403
Deploy Aion BRE Web Service Components on UNIX/Linux	405
Prerequisite	405
Prepare for WebLogic Deployment	405
Prepare for WebSphere Deployment	406
Deploy the Aion BRE Application	408
Code Generation for Web Service Deployment	409
Client Programming Considerations	410
Administering Aion-Based Web services	411
Additional Resources on Web Services	411

Chapter 18: Debug Aion BRE Applications 413

Debugger Features	415
Embedded Component Debugging	416
The Debugger Window	418
The Debugger Toolbar	418
Stack List Box	419
Arguments Pane	420
Watched Attribute Pane	421
Method Body Pane	421
Instance Counter	422
Debugger Tab Pages	422
Breakpoints	429
Data Breakpoints	429
Code Breakpoints	430
Set and Remove Data Breakpoints	430
Set and Remove Code Breakpoints	431
Watchpoints	432
Set and Remove Watchpoints	432
Debugger Settings	432
Configure Debugger Settings	432
Debug Aion BRE Applications	433
Set Breakpoints and Watchpoints	433
Control the Flow of Execution	433
View and Modify Data Values	434
Use the Call Stack	435
Debugging Rule-Based Inference	436
Backward Chaining	437
Forward Chaining	437
Special Considerations for Decision Tables	437
Shut Down the Debug Session	438

Chapter 19: Run and Build Applications 439

Run Aion BRE Applications	439
Run Aion BRE Applications Interpretively	439
Run Stand-Alone Applications	439
Compile Applications	440
Prepare to Build Aion BRE Applications	441
Configure Build Settings	441
Configure Library Properties	442
Build the Application	443
Stop the Build	443

Use Interface Layers	444
Select an Interface Layer	445
Deploy the Application	446

Appendix A: CA Aion Business Rules Expert with UML Modeling Software **447**

Object Types	447
Transferred Properties	448
Associations, Association Ends, and Pointers	449
Associations in UML	449
Association Classes	450
Map Association Ends to Pointers	450
Import and Export UML Models	451
Import UML Models	452
Merge UML Models with Aion Applications	452
Export to UML Models	454
CA Component Modeler	454
Data Type Mapping	455
Modeling _Interfaces	456
Standards	456

Appendix B: Aion--Acknowledgements **459**

Apache Software License, Version 2.0	460
Apache Software License, Version 1.1	464
BusinessObjects Software License	467
CA Trusted Open Source License	480
HP Java 2 Runtime Environment 1.4.2	488
HP-UX Runtime Environment for the Java 2 Platform, Version 1.4	490
IBM 32-bit Runtime Environment for AIX, Java 2 Technology Edition, Version 1.4	492
IBM Java 2 Runtime Environment 1.3.1	496
IBM zSeries Developer Kit for Linux, Java 2 Technology Edition	501
ICU License	505
ImageMagick Studio	506
Java 2 Runtime Environment (J2RE), Standard Edition, Version 1.4.x	508
Java Naming and Directory Interface (JNDI), Version 1.2.1	510
Java XML Pack Summer '02 Bundle	512
JavaBeans Activation Framework, Version 1.0.2	514
JavaMail, Version 1.3	516
Sun Microsystems, Inc. Java 2 Runtime Environment 1.4.2	518
JUnit 3.8.1	521
OpenSSL 0.9.7c	527
Sun Microsystems, Inc. JIMI SDK, Version 2.0	530

TPSR P05273_11	532
Werken digital SAXPath 1.0	538
Werken Company Jaxen 1.0	540

Index **543**

Chapter 1: Introduction

CA Aion BRE provides an integrated development environment that allows organizations to build intelligent components and applications that can be flexibly deployed throughout the enterprise. Combining business rules technology and object-oriented programming, CA Aion BRE allows companies to build maintainable applications that capture the knowledge and expertise of their organizations. The term Aion BRE refers to the technology of CA Aion BRE. Aion BRE applications can be deployed as distributed components across a range of enterprise architectures and platforms.

This guide describes the CA Aion BRE facilities for prototyping, creating, and testing knowledge-based applications. Context-sensitive help also provides links to CA Aion BRE online help for step-by-step instructions for using these facilities.

Note: Unless otherwise indicated, the term *Windows* refers to any Microsoft Windows operating system supported by CA Aion BRE, including Microsoft Vista, Microsoft Windows XP and Microsoft Windows 2003. Please consult the CA Aion BRE Readme (Operating System Support) for the service levels of each operating system that are supported.

This section contains the following topics:

[Rules and Inferencing](#) (see page 19)

[Full Object Orientation](#) (see page 20)

[Component-Based Development](#) (see page 21)

[Multiple Application Architectures](#) (see page 21)

[Graphical User Interface Builder](#) (see page 22)

[Visual Editors and the Method Editor](#) (see page 22)

[Aion BRE Language](#) (see page 22)

[Database Support](#) (see page 23)

[Rapid Application Development](#) (see page 23)

Rules and Inferencing

In CA Aion BRE, knowledge is represented by rules. In general, rules are simple, non-procedural if-then statements, written without consideration of control flow. Rules test and make assignments to attributes of the business objects that are the focus of your application. You write rules in method bodies.

An inferencing command occurs in a method body. It passes control to the inference engine, which reasons-draws logical conclusions-about the business data by processing the rules. CA Aion BRE supports four kinds of rule-processing:

- Forward chaining: the engine runs data through all rules to see the consequences.
- Backward chaining: starts with a goal, which is an attribute set by one or more of the rules. The engine only uses rules that lead to conclusions about the goal attribute.
- Pattern matching: a simple and powerful way to iterate over the instances of one or more classes.
- Truth maintenance: allows “what-if” scenarios to be run, in which you can tentatively set attribute values, determine their consequences, and then retract the values.

Rules and Inferencing are discussed in detail in the *CA Aion BRE Rules Guide* and CA Aion BRE Rules online help.

Full Object Orientation

Development in CA Aion BRE is completely object-oriented. All aspects of an application from the user interface to data access are created from the class libraries shipped with CA Aion BRE. You can use these supplied classes directly, or you can create new ones by subclassing.

Rules are also treated as objects in CA Aion BRE. This enables rules to benefit from the protection of data encapsulation and reuse made possible by inheritance and polymorphism. A single programming paradigm can carry you throughout the development process.

More information:

[Overview of CA Aion BRE Objects](#) (see page 43)

Component-Based Development

You can partition your Aion BRE application into components. Separating the application into components for the graphical user interface, database access, and knowledge base (the rules) can speed up development. A different developer or team can work on each component.

Component-based development also means you do not have to start from scratch with each new application. Any Aion BRE component or an entire application can be included in a new application to streamline development time.

You are not limited to using components created in CA Aion BRE. You can plug in Active-X controls and access their methods from your Aion BRE application. If you have C or C++ DLLs you want to use, you can access functions exported from them, too.

Aion BRE applications can also be used by foreign components. You can expose exported methods by wrapping the Aion BRE component in an interface layer. You build the component as a COM, Java, C, or C++ object by choosing the appropriate interface at build-time.

Components are discussed in the following chapters of this guide:

- Generate and Use C and C++ Components
- Generate and Use Managed C++ Components
- Generate and Use Java Components
- Generate and Use COM Components

Multiple Application Architectures

You can create Aion BRE components for a variety of client-server architectures. For example, you might place the database access component on one server, the rules on another, and the graphical user interface on a third. You have your choice of interfaces to wrap the components: C/C++, Managed C++, Java, and COM. Furthermore, you can access existing foreign components, using their exported methods or exposing Aion BRE methods to them. All of these object components can communicate with Aion BRE components across a LAN, WAN, IBM MQSeries queues, and the Internet. Under the Web, CA Aion BRE supports J2EE servlet technology.

Graphical User Interface Builder

CA Aion BRE offers state-of-the art facilities to create a graphical user interface (GUI) for your application. You add and edit controls visually by dragging their icons onto a window or dialog box and then moving, resizing and otherwise modifying their properties. Supported controls include rich edit windows, multi-column list boxes, tree controls, image lists, and tab pages.

More Information:

[Create a Graphical User Interface](#) (see page 139)

Visual Editors and the Method Editor

You create an application using various visual editors. Each object type has its own editor. For example, you modify class properties in the Class Editor and instance values and properties in the Instance editor. The Window, Menu, Toolbar and Tab editors let you visually assemble and edit a GUI. Database access is handled through the Stored Procedure and Query editors. CA Aion BRE provides a Rule Editor and a Decision Table editor for modifying knowledge objects.

CA Aion BRE automatically generates code from the visual editors, but you also can modify Aion BRE code programmatically in the Method Editor. Anything the visual editors do can be coded manually in the Method Editor. Sometimes you will use a visual editor and the Method Editor in tandem, as when you right-click a control in one of the GUI editors and use the Method Editor to input an event handler for each event you wish to define. The Method Editor displays code in customizable colors, and visually distinguishes keywords, string literals, comments, and so forth.

Aion BRE Language

CA Aion BRE has an easy-to-learn object-oriented language for writing methods (including both procedural logic and non-procedural rules). Procedural logic is written in this language. CA Aion BRE expresses non-procedural logic through the use of INFER block constructs.

The Aion BRE language is described in the CA Aion BRE online help.

Database Support

An Aion BRE application can connect to a database and retrieve data. You define SQL queries and stored procedures in CA Aion BRE by subclassing the supplied classes. Each record of the result set is an instance of that subclass.

Aion BRE supports native drivers for Oracle10g, Oracle11g, Sybase ASE 12.5, Microsoft SQL Server 2005 and 2008, and IBM DB2 for LUW 8.1 and 9.1 and IBM DB2 for zOS 8.1 and 9.1 as well as an ODBC driver.

Note: To assure that DB2 database access is successful the user must use the same version of DB2 Client and Server.

Rapid Application Development

The CA Aion BRE development environment features a comprehensive set of development tools, including a graphical debugger and extensive tracing facilities. In addition to standard debugger behavior, the Aion BRE Debugger lets you step through the execution of rules and view the state of any given rule.

Aion BRE applications run in interpreted mode within the integrated development environment (IDE), providing immediate feedback during the development process. At any point during the development of a standalone application, you can run it interpretively, or you can fully build it into an executable, using a C++ compiler from within Aion.

More Information:

[Run and Build Applications](#) (see page 439)

Chapter 2: Installation and Setup and Uninstall

This section contains the following topics:

- [Installation Prerequisites](#) (see page 25)
- [Install on Microsoft Windows](#) (see page 26)
- [Uninstall on Microsoft Windows](#) (see page 30)
- [Install on Linux/UNIX server](#) (see page 31)
- [Uninstall Linux/UNIX server](#) (see page 39)
- [Silent Installation](#) (see page 40)

Installation Prerequisites

Before you begin any of the procedures in this chapter, be sure to do the following:

- Review the *CA Aion BRE* Readme.

This document contains late-breaking product news and information about installation considerations, system requirements, hardware requirements, operating systems supported, pre-installation requirements, third-party software compatibility, known issues and how to contact CA Technical Support.

- Privileges required

In order to install CA Aion BRE on Microsoft Windows Vista or Microsoft Windows 2003, you must log onto Microsoft Windows as a user with administrator privileges. On UNIX/Linux, you must log on as a user with root privileges if installing into system directory that is /opt/CA.

- For the required space that will be used in the Linux/Unix /tmp folder you should specify 300MB or more.

Install on Microsoft Windows

CA Aion BRE release 11.0 is a multiplatform application.

To install CA Aion BRE, follow this steps:

1. Obtain the general multiplatform installers link and go to Windows platform and find VM directory.

Note:

- You can change the default install directory.
- You will see the installation log file, `aionbre_install.log`, if you copy DISK1 to your local directory and then install. If you install from the remote sever the `aionbe_install.log` will not generate. This is a known issue that can be found in the *CA Aion BRE Readme* file.

2. Double click on `setup.exe`.

The install wizard starts and the InstallAnywhere preparing to install screen displays.

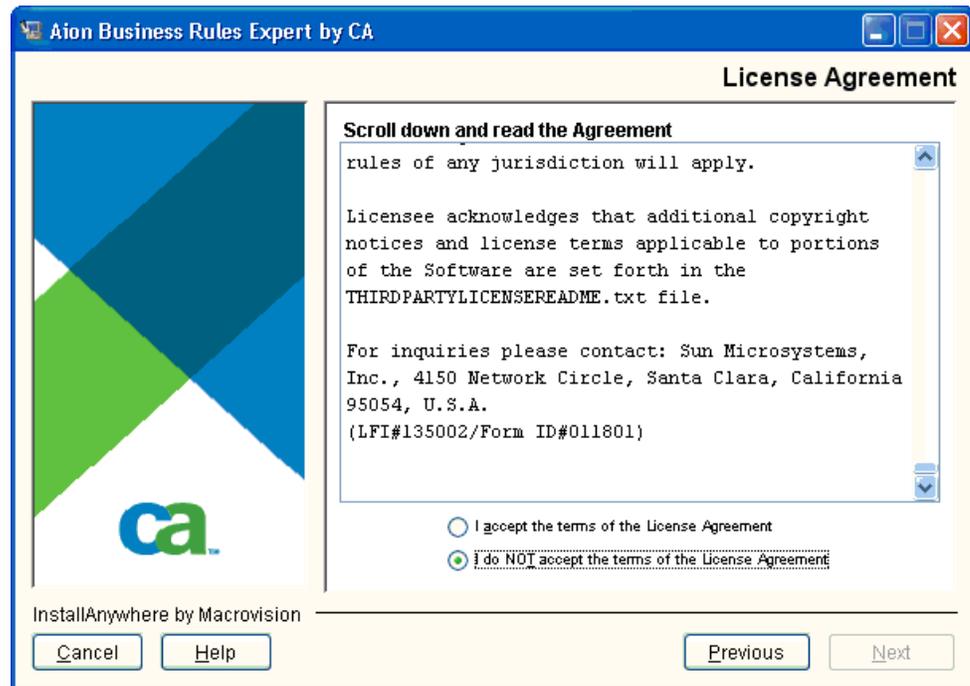
When the prepare install completes the Introduction screen displays

3. Read the important information on the Introduction screen.
4. Click next.

The License Agreement screen displays.

5. Read the entire License Agreement, select I agree, and click Next.

Note: If you do not agree with the terms of the License Agreement, you must click I do NOT accept and the installation process ends.



The Choose Install Set screen displays.

Select the type of installation. Typical is the Default.

6. If you selected Typical, click Next.

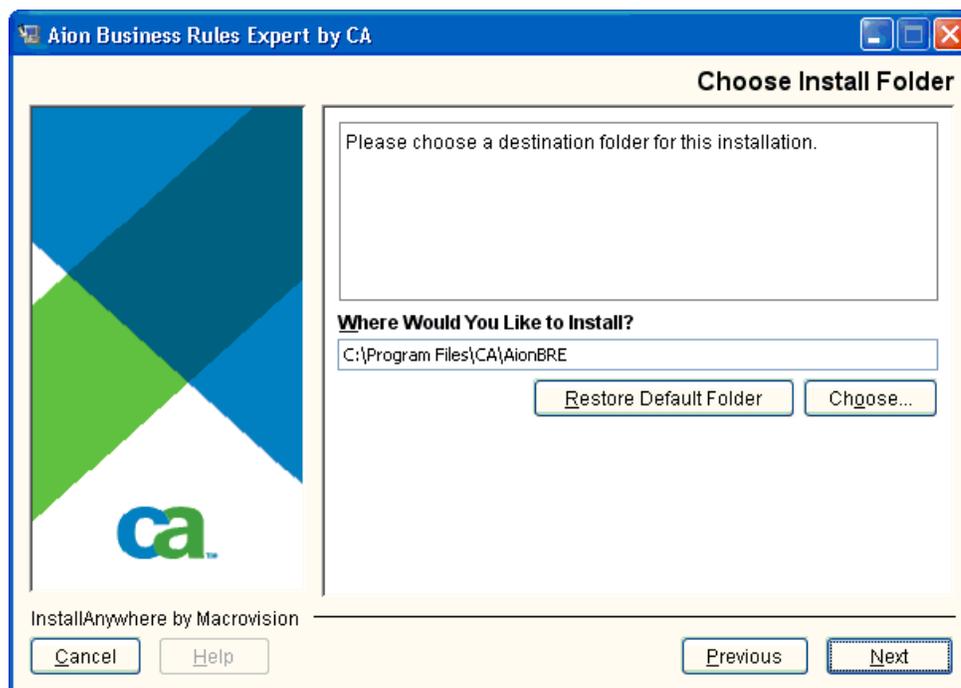
Note: The Typical install will install the following features (ASCII) to the system

- Aion BRE Dynamic Rule Manager
- Aion BRE Dynamic Rulebase Administrator
- Aion BRE Dynamic Integrated Development System
- Help

The Choose Install Folder screen displays.

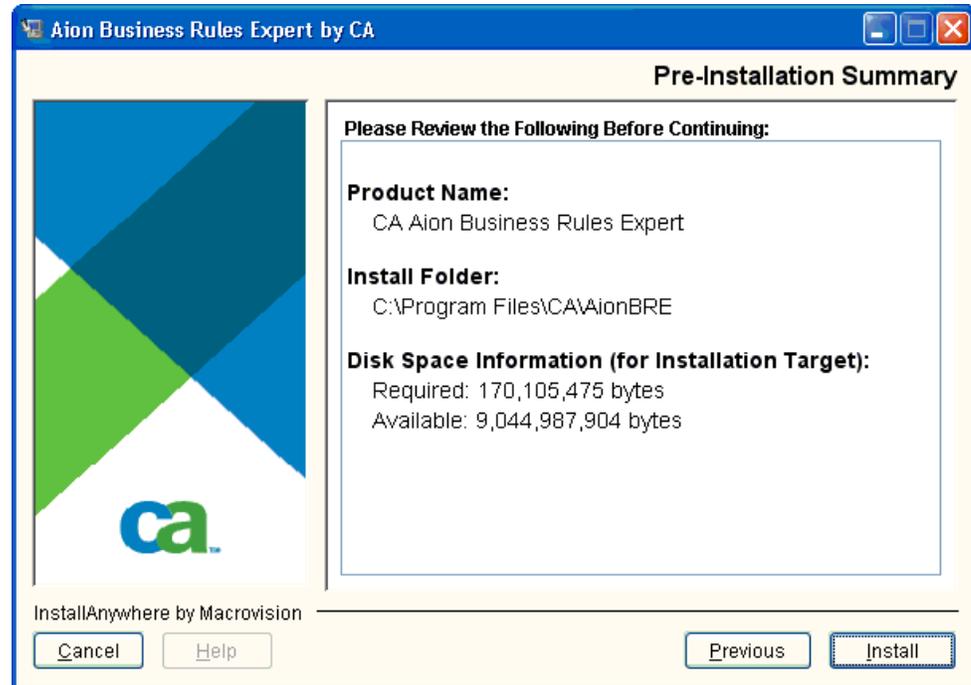
Note: You can change the default install directory.

7. Click Next to accept the installation path, or click Choose... to select a different installation directory.



The Pre-Installation Summary screen displays.

8. Click Install.



The Installing CA Aion Business Rules Expert screen displays.

The Install Complete screen displays.

9. Click Done.

Custom Installation

In the Custom installation, you can choose the desired features with an option of Unicode or ASCII (non-Unicode). After you choose Custom install set on the Choose Install Set panel, click Next. The panel of Choose Install Features will be displayed. By default, all features are unchecked:

- Aion BRE Integrated Development System (IDE)
- Aion BRE Execution System (RUNTIME)
- Aion Advanced Builder Option (COBSLIB)
- Aion BRE Dynamic Rule Manager (DRM)
- Aion BRE Examples
- Help

After you select the feature(s), click Next. The panel of Choose Install Folder will be displayed. Make a change or take the default location and click Next. The panel of Choose The Type Of Executables will be displayed. The user can choose to install Unicode executables or ASCII (non-Unicode) executables. Make a choice and click Next. In the Pre-Installation Summary panel you have a final opportunity to review all necessary information about this install. Click on Install button to start.

Note: The feature DRM will install both Dynamic Rule Manager and Dynamic Rulebase Administrator.

If you installed ASCII executables of Aion BRE and later decide to switch to Unicode, you must first uninstall ASCII and then install Unicode executables, and visa verse.

Uninstall on Microsoft Windows

To uninstall CA Aion BRE on Microsoft Windows, follow these steps:

1. Click Start and go to Settings, Control Panel, Add or Remove Programs.
2. Select CA Aion Business Rules Expert.
3. Click Change/Remove.

The uninstallation wizard removes CA Aion BRE from your system. You have options of uninstalling entire product or uninstalling features.

Note: You can also uninstall AION BRE using `uninstallaionbre.exe` from Aion BRE HOME directory

4. After uninstall you need to reboot your system.

Install on Linux/UNIX server

Console-Mode Installation

Obtain the AionBRE r11 installation program for Linux/Unix platform for installation.

To start the install CA Aion BRE for Linux/Unix with Console-Mode, follow these steps:

1. Log in to the target Linux/Unix system with root privileges.
2. Go to the VM directory where you downloaded the installation program.
3. Launch the installation by entering the following command:

If root log in

```
sh ./setup.bin -i console
```

If regular user log in with sudo access right

```
sudo sh ./setup.bin -i console
```

To complete the Console-Mode installation procedure, follow these steps:

Note: For each section in the install, enter the number associated with your choice or by press Enter to accept the default.

1. Introduction prompt, press Enter to continue the installation process.
2. License Agreement prompt, please read the entire agreement and indicate your acceptance or rejection of its terms.
3. Type "Y" to accept, the installation process will be continued.
4. Type "N" to reject, the installation process will be terminated.
5. Choose Install Set prompt, press Enter (currently only Typical available).
6. Choose Install Folder prompt.
7. If accept default install folder as /opt/CA/AionBRE, press Enter.
8. If instead of default folder with another install folder, please input an absolute path then press Enter.
9. Pre-Installation Summary prompt, press Enter to continue process.
10. Ready To Install prompt, press Enter to continue process.
11. Installing prompt, it will take a few minutes to complete.
12. Installation Complete prompt, press Enter to exit the installation program.

Graphical-Mode Installation

To run graphical-mode installation, make sure your current login session must support the Java-based GUI, if the installation detects the current session doesn't support Java-based GUI, the error will display and installation be terminated.

To start the install CA Aion BRE for Linux/Unix with Graphical-Mode, follow these steps:

1. Log in to the target Linux/Unix system with root privileges.
2. Go to the VM directory where you downloaded the installation program
3. Launch the installation by entering the following command

If root log in

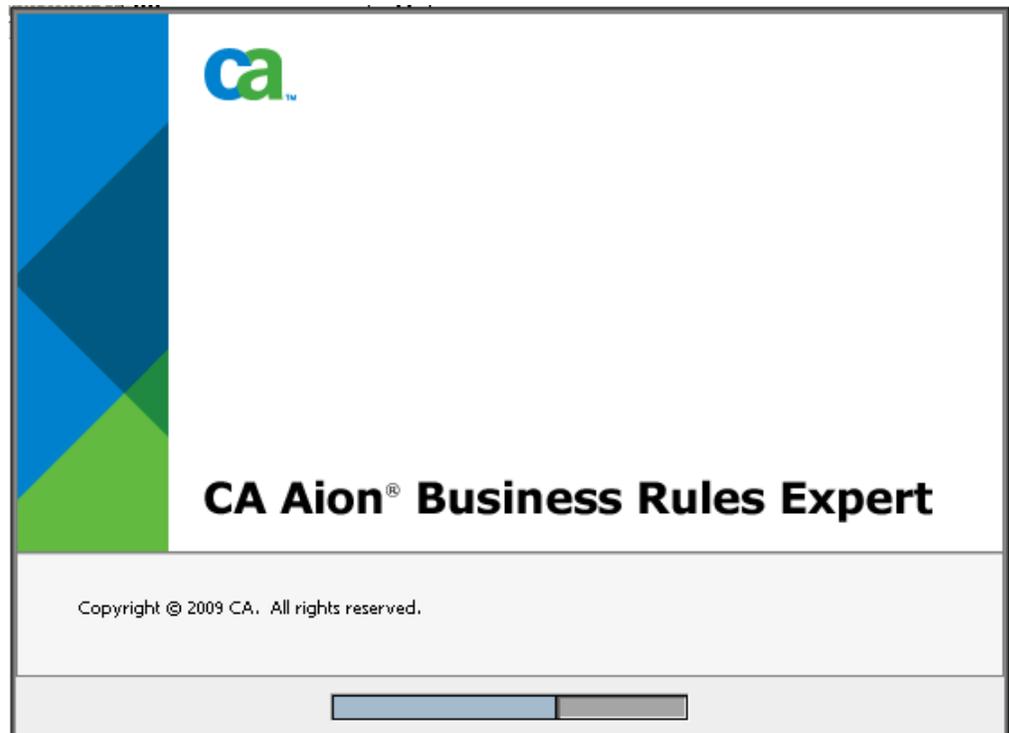
```
sh ./setup.bin
```

If regular user log in with sudo access right

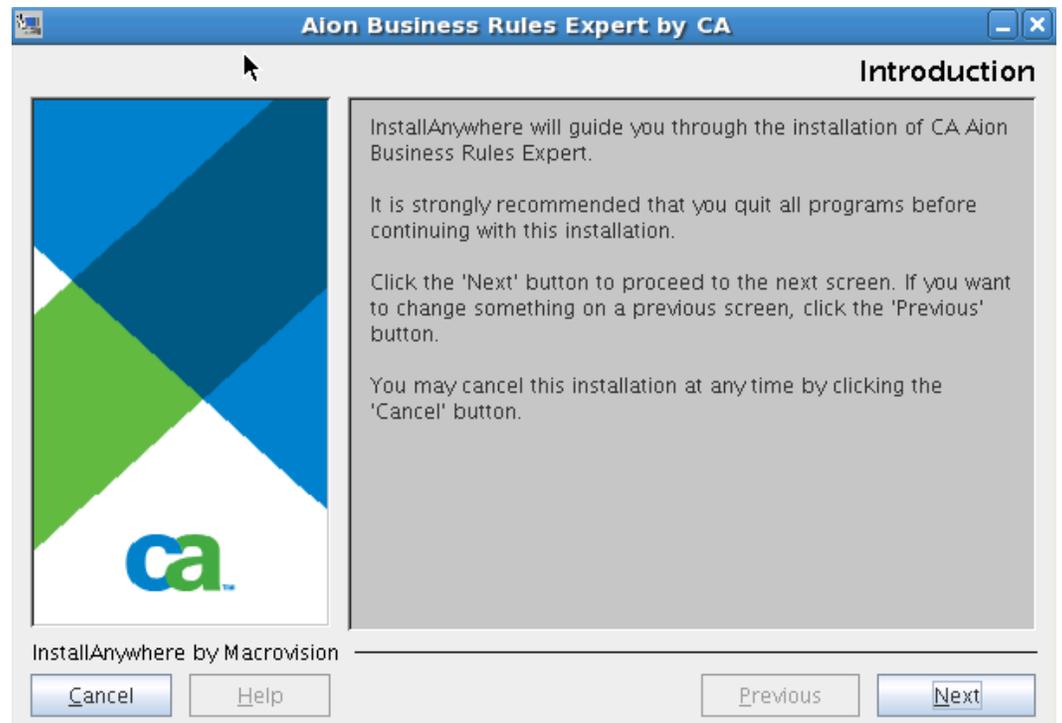
```
sudo sh ./setup.bin
```

To complete the Graphical-Mode installation procedure, follow these steps.

1. Launching Installation program



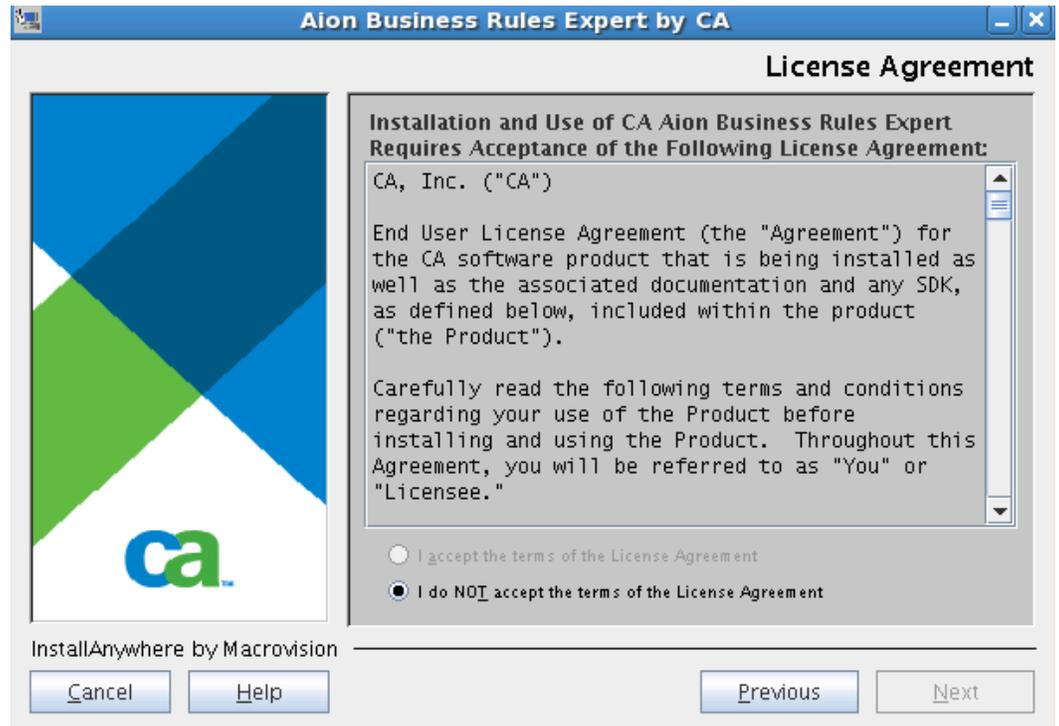
2. Introduction displays, read the important information on the Introduction screen.



3. The License Agreement screen displays.

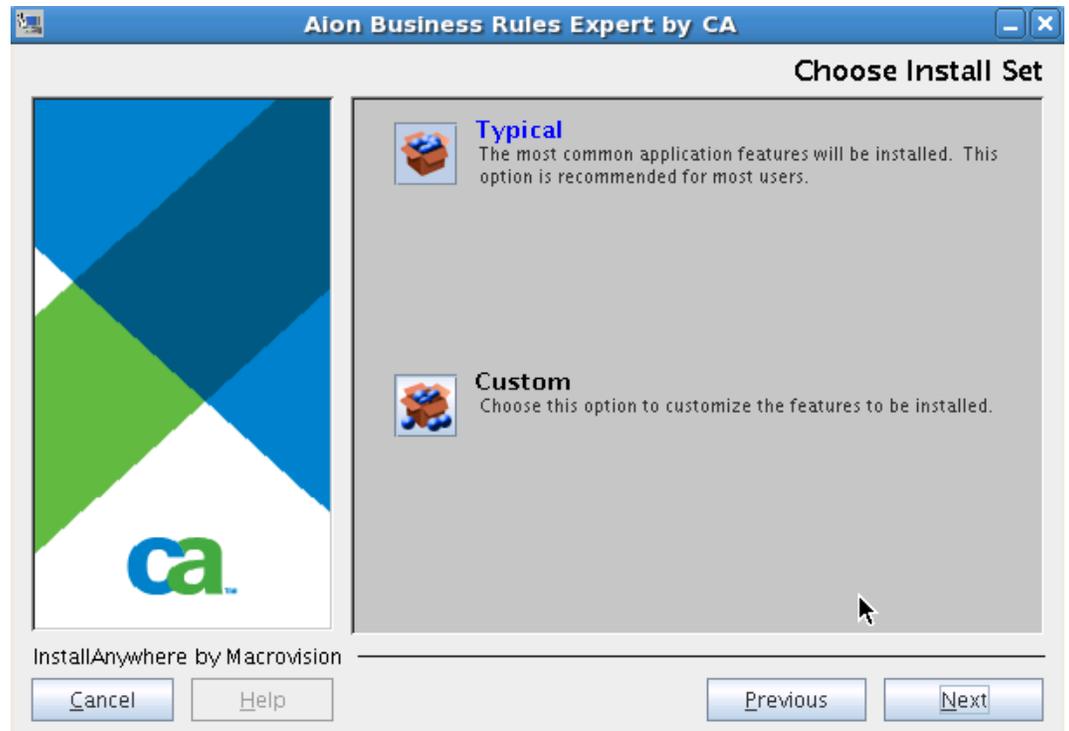
Read the entire License Agreement, select I agree, and click Next

Note: If you do NOT accept the terms of License Agreement, the installation process will be terminated.



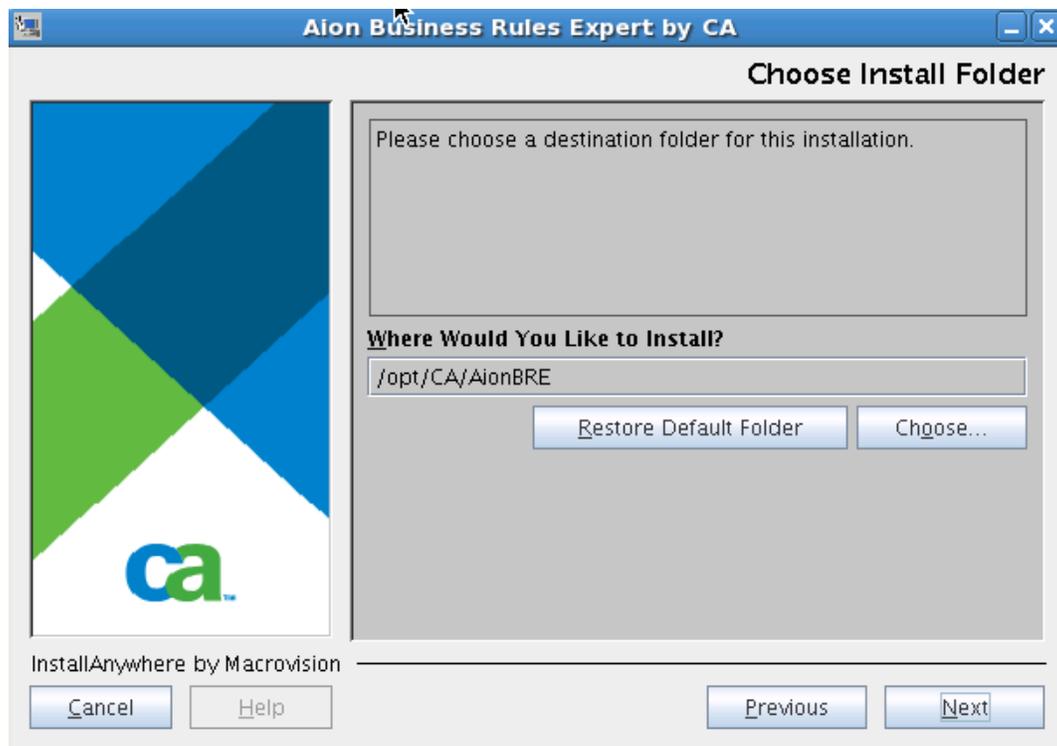
- The Choose Install Set displays. Choose "Typical" and click Next.

Note: There is no different between the Typical and Custom.

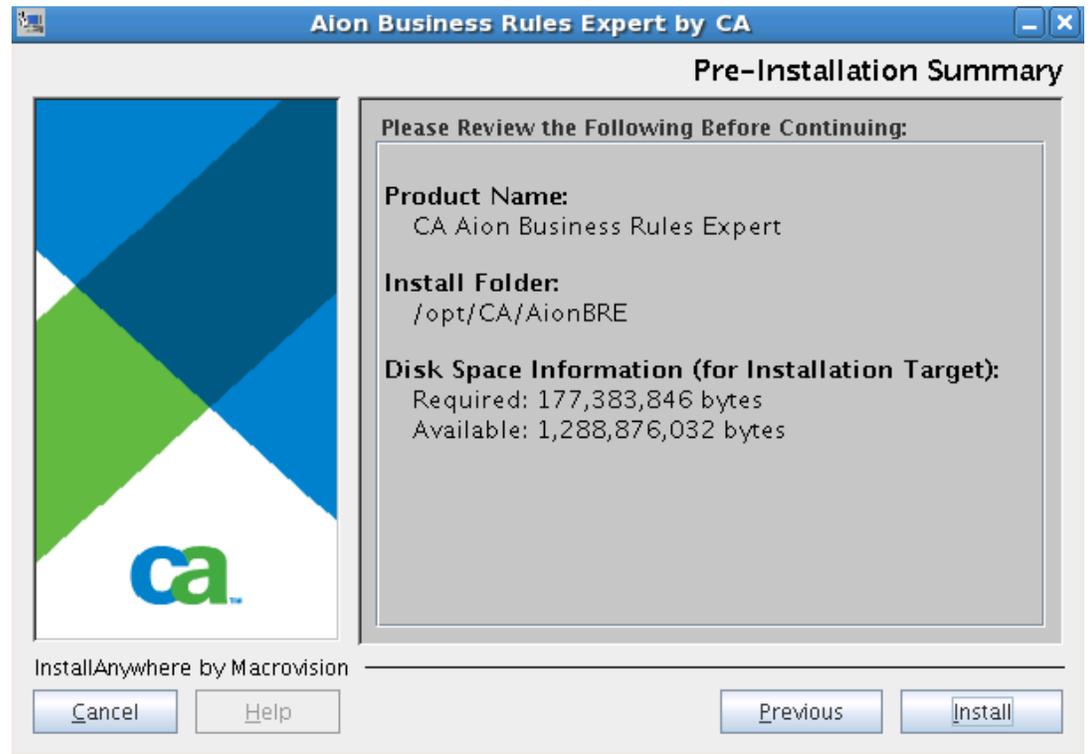


5. The Choose Install Folder displays.

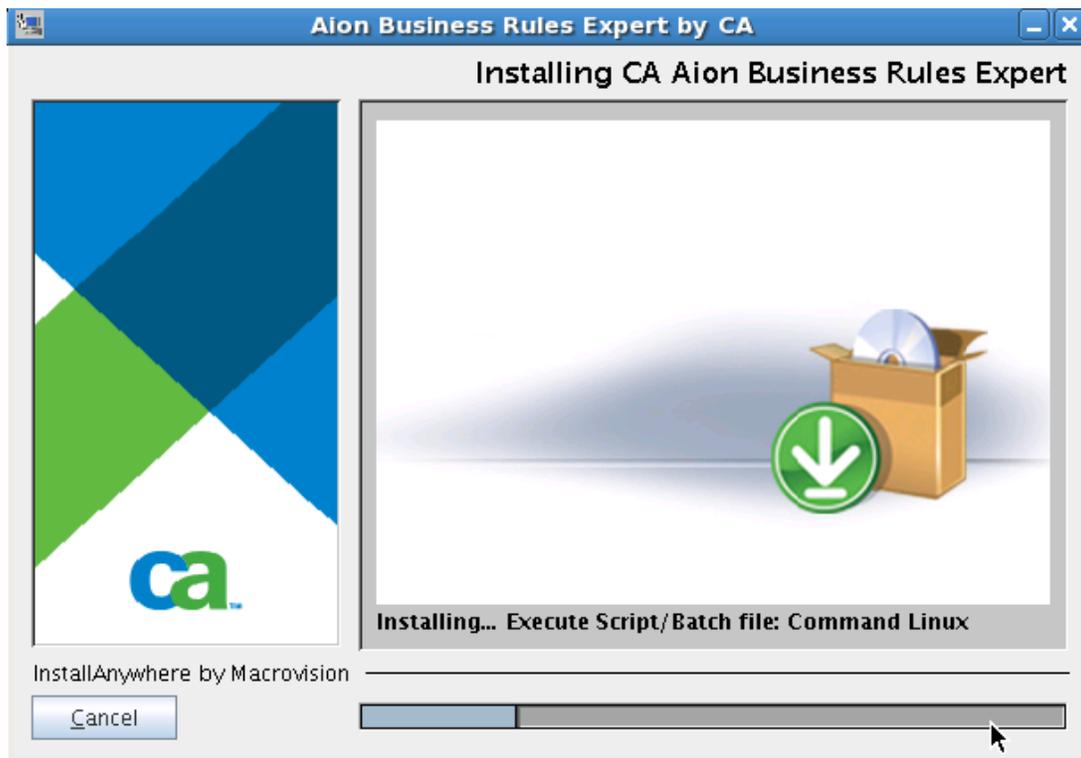
You may click Choose... to select your specific install folder other than default install folder /opt/CA/AionBRE



6. The Pre-Installation Summary screen displays, click Install.



7. Installing progress displays.



8. The Install Complete screen displays, click Done to exit installation program.



Uninstall Linux/UNIX server

If your session supports a graphical user interface, the uninstall program will start in graphical mode, otherwise, it will start in console mode.

To uninstall in the console mode, follow these steps:

1. Go to `$AION_HOME/Uninstall_AionBRE` directory.
`$AION_HOME` represents the directory in which you installed Aion BRE
2. Enter `./Uninstall_CA_Aion_Business_Rules_Expert` at prompt

Console-Mode

Uninstalling... is displayed.

It'll take less than 30 second to complete uninstall Aion BRE

To uninstall in Graphical-Mode, follow this step:

1. Follow the uninstallation wizard to complete uninstall Aion BRE

Important! The uninstall process will not delete files that have changed since the install. This includes any applications that have been rebuilt from source. Directories that contain files that have changed will not be deleted.

Silent Installation

Silent mode which enables an installer to run without any user interaction, is fully supported on Windows and all UNIX platforms. Silent Installation may be down with default setting defined in the installer or with a respond file.

To Silent Installation with default settings defined in the installer, follow these steps:

1. The installer performs the Typical type install and install the product to the directory \Program Files\CA\AionBRE.

For Windows, issue the command:

```
setup.exe -i silent
```

For Unix/Linux, issue the command:

```
sh ./setup.bin -i silent
```

To Silent Installation with a response file, follow these steps:

1. The installer retrieves the values for various InstallAnywhere variables used to control the install from the response file. The response file can be named liberally. The example of the typical content of the response file follows:

For Windows:

```
INSTALLER_UI=silent  
USER_INSTALL_DIR=C:\\Program Files\\CA\\AionBRE
```

For Unix/Linux:

```
INSTALLER_UI=silent  
USER_INSTALL_DIR=/opt/CA/AionBRE
```

2. To do silent install with response file, type the following command with the direct or relative path to the properties file:

For Windows, issue the command

```
setup.exe -f myresponse.properties
```

or

```
setup.exe -f C:\mypaths\myresponse.properties
```

For Unix/Linux, issue the command

```
sh ./setup.bin -f myresponse.properties
```

or

```
sh ./setup.bin -f /home/installer.properties
```

Note: The installer generates its own properties file `installer.properties` in both Graphical-Mode and Silent mode installs. The better practice is to make your own properties file with different name.

Chapter 3: Overview of CA Aion BRE Objects

This chapter contains a high level overview of Aion BRE objects.

CA Aion BRE is a fully object-oriented development environment. A single class hierarchy provides all components of an Aion BRE application knowledge bases, user interfaces, and database access. When creating an application, it is typically convenient to work with class libraries classes grouped together because of the functionality they provide. Libraries streamline application development and permit the efficient re-use of code.

This section contains the following topics:

- [Applications](#) (see page 43)
- [Object Orientation](#) (see page 47)
- [Anatomy of a Class](#) (see page 63)
- [Dynamic versus Static Instances](#) (see page 70)
- [Implement Interfaces](#) (see page 72)

Applications

An Aion BRE application is object-oriented and consists of the following elements:

- Classes
- Instances of classes
- Methods and attributes that make up classes
- Constants

Aion BRE-supplied applications (such as SysLib) are called **system libraries**. Applications developed using Aion BRE are called **custom libraries**. An Aion BRE application can include system libraries and custom libraries.

Every class in an Aion BRE application ultimately derives from the `_system` class defined in the Aion-supplied (and automatically included) SysLib library, which means Aion BRE application is a single class hierarchy. When creating an application, however, this view of the class hierarchy is of little use. A programmer needs to know which classes work together to produce the desired functionality.

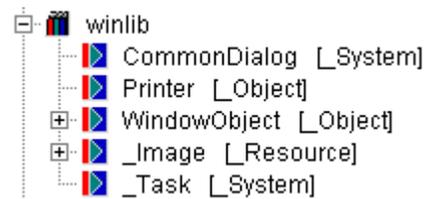
Note: If you are unclear about object-oriented terms like *class* and *instance*, skip forward to the Overview of Object Orientation section.

Libraries Are Functional Units

In order to provide a more helpful view of classes, CA Aion BRE uses class libraries-collections of classes grouped by the functionality they provide. The classes of a library need not be related by inheritance or reside on the same branch of the class hierarchy.

Consider WinLib, an Aion-supplied library with five classes at the top level, including `CommonDialog` and `_Image`. The fact that they are grouped on the same level does not imply that they are sibling classes.

The following illustration shows WinLib classes:



For instance, the base class of `CommonDialog` is `_System`, while the base class of `_Image` is `_Resource`. The criterion for inclusion in WinLib is whether a class enables the creation and display of standard windows, dialog boxes, and controls, not where it falls in the single class hierarchy.

Of course, each top-level class in WinLib has several levels of subclasses that are also part of WinLib. Consequently, the classes in WinLib form several trees, and within a tree the classes *are* related by inheritance.

An Aion BRE developer does not need to know the distant ancestry of the class being used. Knowing the base class one or two generations above your class is usually adequate. Within a library, the whole class hierarchy is not always visible. Instead the developer works from an alphabetical list of classes.

Included Libraries

To access classes and instances in a System Library or custom library, you must include the library in your application. After you do so, you can view and subclass the included library's classes.

An application consists of its own library (the application, itself) and any included libraries (SysLib, at a minimum). If an included library, itself, contains an included library, including the outer library means you automatically get the nested library, too. For example, the system library IOWLib contains another system library named IOLib. If you include IOWLib, your application automatically includes IOLib as well.

Aion system libraries provide support for a wide variety of functionalities, including user interfaces, database connections, data access, report creation, and generation of object communication layers.

In addition to Aion BRE system libraries, you can include any custom Aion BRE application as well. This permits re-use of existing Aion BRE code.

Stand-Alone Aion BRE Applications

A *stand-alone* Aion BRE application is one that can be run (as compiled or interpretively) without having to be called or included. A stand-alone application contains a class method named `Start` in a class that has the *Entry Class* property checked. Typically (although not necessarily), this entry class is called `Main`. For an application to be stand-alone, it must have only one entry class and `Start` method, and the entry class must reside in the application itself, not an included library or application.

Note: When you include a stand-alone application in another Aion BRE application, CA Aion BRE ignores the `Start` method in the included application.

Once an application is started, all subsequent execution is controlled by message passing between classes and instances.

Benefits of Included Libraries

The main reason for creating and including modular application components is to partition a project into logically-related, manageable units. Two primary benefits of partitioning an application are:

- Different developers or teams of developers can work on different libraries simultaneously without hindering each other's work
- You can reuse a library in other applications.

You can facilitate team development by dividing an application into libraries and placing it under source control. A developer modifies a library by checking out a copy to a local drive. When changes have been successfully incorporated and tested, the developer checks the library back in. You can more easily track modifications because they are modular. Each set of changes affects only one version of one library.

Partitioning an application into libraries facilitates code reuse as well. For example, you might divide an application's main business processing into a collection of applications. You could reuse the well-designed logic in several applications, while creating separate GUIs for each application. Included libraries promote consistency and economy in the development process.

You can also employ code re-use to promote a common look and feel for all applications across an enterprise. Simply isolate your specialized GUI classes in a custom library of GUI elements.

Library Boundaries

A base class can belong to one library while its subclass belongs to another. Thus, an included library has boundaries, both upper and lower, where it is contiguous with other libraries. Library boundaries dictate how you edit a library as well as how you access its classes and instances.

Edit Libraries Across a Boundary

You cannot edit an included library directly from an application. The application in which it is included has read-only access to the library's classes and instances. If you wish to edit classes of an included library, you have two choices:

- Open the library from within the IDE. Aion BRE closes the application you were working on and opens the library for editing.
- Subclass the class you wish to use. The subclass is part of your application library and you can modify it as needed. This process does not modify the included library.

Create Instances Across a Boundary

Aion BRE allows you to create instances from a class defined in another library. For example, the *Connection* class is defined in the DataLib. When you establish a database connection for a project, a static instance is automatically created in your application library. Similarly, dynamic instances of simple GUI controls, like check boxes and radio buttons, can arise directly from the WinLib classes without subclassing.

For most purposes, however, this approach is too limiting. A class someone else wrote rarely has exactly the behavior and data structure you desire.

Subclass Across a Boundary

This is a technique you will use often, as it provides great flexibility. Some classes you must subclass before using them. An example is any container class in WinLib, such as the DialogBox class, StandardWindow class, or MenuItem class.

In other situations, you can use a class directly by instantiating it in your application, but you are better off subclassing first. For example, it is best to subclass the CheckBox or RadioButton class before using either one in your application.

Subclassing not only allows you to specialize a class, but to create a custom library as well. In the example mentioned previously, you can create a custom library of your company's GUI classes to standardize the look and feel of applications without having to re-code. This approach, however, requires that you subclass *all* classes that are to be part of the custom library.

Object Orientation

Object orientation is a powerful conceptual framework for modeling relationships between entities. Applicable to diverse domains, it has become one of the central paradigms of software development. Object orientation can model the business data in a program, and it can model relationships between modules of code as well.

Since object-oriented theory and practice is such a vast field, this section seeks only to outline object-oriented terms and concepts used in CA Aion BRE. For further information on object orientation, you can find many fine books in the computer section of your bookstore.

Basic Object-Oriented Terms

Objects model things in the world, such as companies, employees, animals, cars, bank accounts, events, and even programming code. An object may be understood as an individual entity that encapsulates a specific set of data—creating an interior where data may be hidden from direct outside manipulation. An object interacts with the outside world through certain behaviors that it advertises publicly. Among these behaviors is manipulation of the data encapsulated in it.

Other objects may request these behaviors by passing the object a *message*. Since the outside world knows only the messages to which an object responds, behaviors of an object can be encapsulated too. The details of how the object handles the message and produces the behavior may be hidden from outside manipulation.

A *class* is like a factory or template for producing objects. A class is composed of a set of behaviors and a set of molds for data. You create one or more objects from a class, using the class as a pattern. Each object has the same behaviors as the class and fills in the molds with its own data values.

In CA Aion BRE, the behaviors of a class or instance are called *methods*. The data holders are known as *attributes*. Together the methods and attributes of a class constitute its *members*. To avoid confusion with the many generic meanings of the word *objects*, Aion refers to them as *instances*. Creating an instance of a class is known as *instantiating* the class.

Apply the Object-Oriented Paradigm

Of the fields to which object theory can be applied, the two that concern developers most are analysis of business data and programming itself.

Apply to Business Data

The previous object-oriented paradigm described can be applied readily to business data. For instance, a Customer class may be instantiated once for each individual customer of a company. An Invoice class yields an individual instance for each purchase. The Customer instances and the Invoice instances encapsulate attributes. They execute methods when sent agreed-upon messages.

Apply to Code

The simplest way to conceive of object-oriented programming is to see attributes as variables (or better yet, as data structures made up of several variables of diverse data types). Methods are functions. Sending a message is just like calling a function.

Object orientation adds the notion of a class. A class combines a certain type of data structure with particular functions. Each instance is a individual data structure holding actual data values. A class or instance can represent software entities such as windows, controls, queries, and database connections.

The data encapsulated in an instance has a special relationship with its methods. In fact, you can protect an attribute from being accessed by functions that are not part of the class. Encapsulation brings much greater data protection than normal structured programming can provide.

Encapsulation includes a class's functions. A calling function defined outside the class need only know the name and parameters of the called function. Implementation details, such as which attributes are used and which other functions called, can be hidden. You can prevent the caller from accessing that data or calling those functions by marking the attributes and methods as private.

The attributes and methods that you make accessible to functions outside the class constitute its *public interface*.

The first essential feature of object orientation, encapsulation, can benefit application programming from GUI design to database access to business logic.

Inheritance

The second essential characteristic of object orientation is *inheritance*, which permits extensive reuse of code within a program. Inheritance defines a relationship between classes in which a *derived class* inherits the methods and attributes of a *base class*. This can be described as an "is-a" relationship; it is the relationship people normally have in mind when they think hierarchically.

For example, a pediatrician *is a* type of medical doctor, which *is a* type of professional. The Pediatrician class inherits from the Medical Doctor class. The Medical Doctor class has attributes such as "medical degree" and methods such as "suture a wound" and "take a temperature." To these, the Pediatrician class adds such attributes as "board certification" and methods such as "administering childhood immunizations."

As you can see, the derived class (also known as a *subclass*) inherits the members of its base class. The derived class can also add new members to those it inherits.

It enjoys another ability as well—that of modifying an inherited member, while preserving the member name. This process is known as *specialization*. The implementation of the class changes, but the public interface remains the same. Returning to our example: a Pediatrician inherits "take a temperature" from Medical Doctor, but may implement it in a specialized way for his patients.

Due to inheritance, modifications you make in a class can be propagated down the class hierarchy, greatly simplifying code maintenance. Inheritance also means that defining new classes is a matter of adding new members to a subclass and specializing inherited members.

Polymorphism

The third essential characteristic of object orientation is *polymorphism*, a word derived from the Greek for *many* and *formed*. Polymorphism is a little obscure, but powerful, making possible full support of specialization. In brief, it enables your code to call a method without indicating which instance or class the method belongs to.

Normally object-oriented programming requires that you specify the particular instance to be manipulated by the called method, using notation that looks like this:

```
InstanceName.MethodName
```

This is known as a *fully qualified* name.

Call Methods of the Same Class

There are occasions, though, in which you would not specify an instance name. For example, when you call a method, it may perform tasks by calling other methods of the same class. You would not want those other calls to specify an instance, since all instances of a class should be able to use the same logic.

Unqualified method calls work fine within a class-until a plain inherited method calls a specialized one. In that case, you risk getting the original method (defined in the base class), instead of the specialized method (defined in the current class) that you actually want.

The principle of polymorphism guarantees that you get the correct method when such calls are made. It does this by introducing the concept of the *current* instance. When an unqualified call is made, a pointer to the current instance is implicitly passed to the called method. As an object-oriented language, Aion takes care of all this for you.

Pass an Instance as an Argument

Another important example of polymorphism occurs when you pass an instance (or pointer to an instance) as an argument to a method. The data type of the argument specifies a class. At runtime the actual instance may be of the specified class. But polymorphism stipulates that the instance may also be of any descendant class of the specified class! As a fully object-oriented language, CA Aion BRE correctly resolves qualified calls within the method body according to the class of the actual instance passed at runtime.

Associations

As powerful as Encapsulation, Inheritance, and Polymorphism are in supporting object-oriented programming (OOP), they are not sufficient to model real world objects, which is the ultimate goal of OOP. Objects exist in a great variety of relationships (for example, mother-of, employee-of, more-expensive-than, and so on). These relationships are called *associations* and they come in an infinite variety and are of varying complexity. Associations are said to have multiplicity, or cardinality, which designates how many of one element can exist in a single occurrence of the association. For example, the association of Company to Employee is said to be a one-to-many association: Each Company can have many Employees. On the other hand, the association of Employee to Company is typically only one-to-one, because an employee can (typically) be employed by only one company.

When programming in Aion, the developer is free to create as many associations between classes as are necessary to model the solution to the problem. An association is (usually) created by defining in one of the associated classes a pointer (for 1-to-1 associations) or a list of pointers (for 1-to-many associations) to instances of the other associated class. For example, to define the Employ relationship between the classes Company and Employee, an Attribute or variable would be defined in the class Company of the type list of pointers to Employee. Similarly, the programmer may wish to define the association Employed-By in which case the class Employee would have Attribute or variable of type pointer to Company.

Defining an association between classes means that instance of one class can access the public methods and attributes of instances of the associated class. Where pAssocClass is an Attribute or variable defined as a pointer to AssocClass, the notation by which a class can access the GetName() method of an instance of AssocClass is:

```
pAssocClass.GetName( ).
```

Note: It is the responsibility of the Aion programmer to create and destruct the relevant associated instances. In an ordinary association, instances are not automatically created or destroyed (compare this situation with the functionality provided by Class Containment). Forgetting to destruct a created instance can cause stranded memory (a chunk of garbage memory that is inaccessible).

Association Classes

Besides the strategy of defining pointers between associated instances, CA Aion BRE provides another way of constructing associations. An Aion-supplied class called `_Association` can be used to define an association as a *thing*. `_Associations` are used when the association being modeled has attributes in its own right that the system needs. For example, the Marriage association (between spouses) typical has such properties as Date and Place. These properties are not easily handled if the instances just have Spouse-of pointers pointing to each other.

Association Classes have the additional advantage of automatically supporting dissolution of the relationship. When associations are implemented as pointers and the instance pointed-to is deleted, there should always a *separate* operation of NULLing the pointer attribute. Failure to NULL the pointer leaves an invalid pointer value. On the hand, if the association is implemented with an instance of a specialized `_Association`, the `_Association` instance is *automatically* deleted when one of its association ends is deleted. There is no more need to worry about cleaning up pointers!

For more information on Association classes, see the CA Aion BRE online help following sections:

- Association Editor
- Using Associations
- Association Class in the "SysLib"

Association classes are illustrated in the Associate and Exasscintf examples provided with Aion.

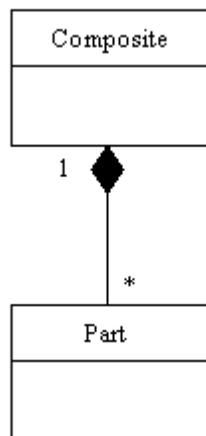
Class Containment

A common association between objects is the *has-a* or *part-of* relationship. This association is so common that it is often represented by a special symbol in OO models. For example, the Unified Modeling Language (UML), which is the industry standard for OO modeling, distinguishes two types of part-of relationships called aggregation and composition, which are represented by an open and closed diamond respectively. Aion offers support for a kind of part-of relationship called Class Containment. Class Containment is, in fact, a more constrained, that is stronger, form of the part-of relationship than what UML calls composition.

In the UML, a composite relationship is defined as:

- The part may be part of only one composite at a time.
- The composite has responsibility for the creation and destruction of its parts.

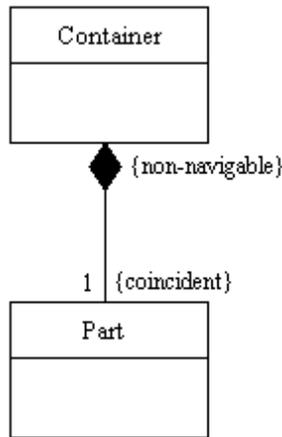
A typical *part-of* relationship involving composition is the relationship between a school and its departments. This relationship is different from a department's relationship to its members (instructors), which is a relationship of aggregation. A school has responsibility for creating and destructing its departments and a school's departments can belong to only that school. On the other hand, instructors are neither created nor destructed by their departments (that is, their *lifetimes* are not the responsibility of their departments), and some schools allow their instructors to be members of different departments at the same time. In the UML, composition is modeled as (where 1 represents the multiplicity of the Part-to-Composite relationship, that is, 1 to 1, and the * (asterisk) symbol represents the multiplicity of the Composite-to-Part relationship, that is, 1 to 0-to-many):



Aion Class Containment goes beyond composition by adding two constraints:

- There can be only one part per unique compositional relationship, that is, the multiplicity of the relationship must always be equal to 1 on the part end of the association. Existence of the part is totally coincident with that the composite, that is, the part is created and destroyed at the same time as the composite is created and destroyed. That is, the *lifetime* of the composite and the part **exactly** coincide.

These additional constraints strengthen the semantics of the Class Containment relationship beyond that of composition. Class Containment may be modeled by means of specific qualifiers on the Part end of the association:



Attribute Level Class Containment

CA Aion BRE supports Class Containment at two levels. The first level is the Attribute level. Class Containment means that the container will have special kind attribute whose value is an instance of the *contained* class. The Aion developer specifies Class Containment at this level by declaring an Attribute's Type to be the contained class in the Attribute Editor. That is, the Type specification for the Attribute is just the name of the contained class rather than *pointer to* the contained class. This declaration means that an instance of the contained class will be created automatically whenever an instance of the container is created and similarly for its destruction. In more technical terms, during the execution of the constructor of a container instance, the constructor of the contained class is invoked, and similarly regarding the container's destructor.

Reference to the (public) methods and attributes of the contained class is accomplished through the Attribute of the container. For example, where an Attribute of the container is `ContainerAttribute`, the value of an attribute of the contained part may be set by invoking its `SetPartAttribute()` method as follows:

```
ContainerAttribute.SetPartAttribute(value)
```

Notice that the same syntax is used as if the `ContainerAttribute` had been declared a pointer to the contained class. Indeed, under the covers, Aion is maintaining a pointer to the contained instance. What Class Containment provides over a pointer is powerful, automatic existence/lifetime control.

Class Containment is especially appropriate when constructing a representation of an object, for example, a form such as an application for a loan, that may have *subparts*, such as an Address part or Financial history part. Often these subparts may require methods of their own in order to specify functionality that is appropriate only for them. Class Containment provides a convenient means of allocating these methods to different objects and thereby not cluttering the main object with many specific methods.

CA Aion BRE employs Class Containment for constructing GUI objects such as Dialog Boxes. Dialog boxes typically contain other GUI widgets such as check boxes, pushbuttons, and text windows. These widgets are defined as instances of GUI classes. When a specific pushbutton is placed on a Dialog Box, a special attribute is created for that particular pushbutton of type `Pushbutton`. Thus, when the Dialog Box itself is created and destructed, all of the contained objects are created and destructed at the same time.

Class Containment for Local Variables

CA Aion BRE extends Class Containment to a second level: local variable declarations. It is possible to declare a local variable, for example., `mylvar`, in a method to be of a type defined by a Class

Example:

```
var mylvar is SomeClass
```

rather than declaring `mylvar` to be a pointer to `SomeClass` (which is typically abbreviated `&SomeClass`). The advantage of using Class Containment for local variable declarations is that an instance of `SomeClass` is not only automatically created when the method begins processing (which saves an explicit `Create()` step) but is automatically destructed when the method finishes. Automatic control over destruction avoids the problem of accidentally leaving memory stranded at the conclusion of method when deleting the instance might easily be forgotten. When a method finishes, the local variable goes away. But an instance explicitly created by invoking a class's `Create()` method will remain in memory as garbage unless explicitly deleted with a corresponding `Delete()` invocation. Using Class Containment for local variable declarations alleviates this concern.

Rules for Assigning Values

As mentioned previously, the same syntax is involved when accessing the methods or attributes of an instance via Class Containment or a pointer. However, there is a difference between these two modes of reference. Consider the following two declarations:

Attribute or variable: HomeAddress is Address

Attribute or variable: pHomeAddress is pointer to Address

Where Street is an attribute of Address, legal assignments are:

```
HomeAddress.Street = "King Street"
```

```
pHomeAddress.Street = "King Street"
```

However, pointer values may be assigned only to Attributes or variables explicitly defined as a pointer.

Example:

The following are legal assignments:

```
pHomeAddress = Address.Create( )
```

```
pHomeAddress = NULL
```

```
pHomeAddress = pOtherAddress
```

```
// where pOtherAddress is of type pointer to Address,
```

these same assignments are illegal for Class Containment. The following assignment will cause an error message in Aion.

```
HomeAddress = pOtherAddress
```

This type of assignment was syntactically permitted in earlier versions of Aion, but it is easy to see why it is an error. The instance to which HomeAddress is pointing would have been changed. This would seem to be permissible because "under the covers" HomeAddress is just a pointer. However, what instance will CA Aion BRE now destruct? It will *not* be the instance that was created in the original declaration, *because reference to that instance has been lost*. The memory allocated to the original instance will be stranded!

Note: Attributes and variables defined with Class Containment can never have the value NULL. For obvious reasons, a Delete() message cannot be sent to a contained instance. Doing so produces a runtime error.

Attached Objects

Besides supporting standard associations, Association Classes, and Class Containment (a specialized association), CA Aion BRE also supports a special relationship called Attachment. The `_Object` class makes available the `Attach()` instance method:

```
Attach(pat is attributePointer, pcl is classpointer to _System = NULL)
```

An object's `Attach()` method can be invoked with either a pointer to an Attribute or a classpointer. The object is then "attached to" the attribute or class. Aion provides special functionality for attached objects.

One very common use of Attachment is to attach GUI controls to appropriate value holders. For example, the following statement attaches `NameBox` (for example, a Text Window) to a value holder (Attribute) identified in the input argument:

```
NameBox.Attach(->valueholder)
```

The value of the `NameBox` control is stored in the value holder Attribute to which `NameBox` is attached. CA Aion BRE automatically retrieves the value contained in value holder for display in the `NameBox` when the GUI control is created on the screen. Furthermore, the `Assign()` method can be used to automatically transfer any data entered into that control on the screen to the value holder to which it is attached.

This use of Attachment is illustrated in the `Attach` example that is provided with CA Aion BRE.

Another, more advanced use of Attachment supports the metaprogramming functionality of CA Aion BRE. For example, an attached object receives "When" events, such as `WhenModified` or `WhenDeleted`, for the class to which it is attached. In other words, attached objects can monitor a class for specific activity to the instances of that class. Through Attached Objects, Aion provides an internal and simple means of implementing a form of the famous Observer pattern (see Erich Gamma, et al., *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley Longman, 1995).

The opposite of the `Attach()` method is the `Detach()` method, which detaches an object from any attribute or class to which it is attached.

More Information:

[Aion Object Event Methods](#) (see page 58)

Aion _Object Event Methods

Event methods are methods that CA Aion BRE automatically invokes when specific types of events occur. The names of these methods follow the form "When*eventType*". Event methods are used throughout CA Aion BRE, especially, in the GUI environment (WinLib), but they are also used to simplify data manipulation (see When... methods for the Data and Query classes). This section addresses the event methods provided by _Object. These methods are:

- WhenCreated
- WhenDeleted
- WhenFlushed
- WhenModified
- WhenSoftDeleted
- WhenSourced

With the exception of WhenSourced, these methods require attached objects through which they are called. For more information on the WhenSourced method, see the section Using the WhenSourced Event Method in the CA Aion BRE *Rules Guide*.

To implement these methods, define an Observer class (subclass of _Object) on which you provide specialized implementations of whichever event method you wish to utilize. Before creating the first instance of the class that is to be monitored, make sure you create an instance of the Observer Class, and attach it to the class being monitored:

```
var p0 is &Observer
p0 = Observer.Create()
p0.Attach(NULL, ClassName)
```

where *ClassName* represents the class to be monitored. The Observer class implements specialized versions of the event methods.

Note: An instance may be attached to its own class so that it can receive event methods. In other words, events methods may be directly implemented on a class when that class is to be monitored by its own instances. Each newly created instance of the class must be attached to the class. However, this approach has limited applicability. For example, it does not support automatic invocations of WhenCreated() and should not be used to support WhenFlushed().

The following table summarizes when these event methods are invoked.

Event Method	When Invoked
WhenCreated()	When a new instance of the class is created

Event Method	When Invoked
	(immediately after invocation of the class's own Create method). Note: The effect of WhenCreated() can also be achieved by specializing the Create() method.
WhenDeleted()	When a new instance of the class is deleted (immediately after invocation of the class's own Delete method). Note: The effect of WhenDeleted() can also be achieved by specializing the Delete() method.
WhenFlushed()	Before instances of a class are flushed, that is., before the Delete() methods of the instances that to be flushed are invoked. WhenFlushed() is invoked only once per invocation of <i>ClassName.Flush()</i> .
WhenModified()	After any attribute of the class is modified. The input parameter specifies the attribute that was modified.
WhenSoftDeleted()	Before a data instance that was loaded from a database is to be deleted. The input parameter specifies what individual is about to be deleted.

For more information on these event methods, consult the CA Aion BRE online help.

Note: In the list of specializable methods for classes derived from `_Object`, you will also see an event method `WhenAttributeModified()`. This is a public class method of `_System` and is to be called only by CA Aion BRE. You should not use this method yourself.

Constraints

CA Aion BRE provides a unique object-oriented approach to defining constraints on attributes. Constraints prevent attributes from being assigned values that are inconsistent with the constraint specification. In CA Aion BRE constraints are defined as **data types** that can be assigned to attributes. For example, a constraint `IntGreaterThanZero` could be defined as a data type specifying an integer greater than zero (0). To apply this constraint to an attribute (for example, `Age`), the programmer declares the attribute to be of type `IntGreaterThanZero`.

Constraint checking is provided for the following types:

- Integer
- Real
- Boolean
- String
- Lists and Arrays of the preceding types

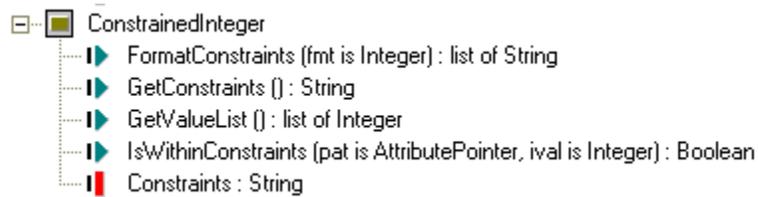
To implement constraints, SysLib provides the following types that are subtypes of Integer, Real, Boolean, and String respectively:

- ConstrainedInteger
- ConstrainedReal
- ConstrainedBoolean
- ConstrainedString

Constrained lists or arrays can be defined to be of type List (or Array) of Constrained*datatype*, where the expression Constrained*datatype* refers to any of these four new data types.

Structure of Constrained Data Types

A constrained data type provides four class methods and one class attribute. For example, the ConstrainedInteger data type has the following structure:



The method `IsWithinConstraints()` is not used frequently by Aion programmers. CA Aion BRE automatically performs testing for compliance with constraints at both edit time and runtime.

There are three ways to retrieve the constraint itself from a `Constraineddatatype`. The `GetConstraints()` method retrieves the constraint expression as a complete string. `GetValueList()` returns the constraint values as a list of values. (However, if the constraint is negative, "is not from", `GetValueList()` returns NULL.) Finally, `FormatConstraints()` returns the constraint expression as a list of strings. These methods offer the Aion programmer a broad set of options for processing constraints, for example, including them in special application output.

Note: These methods are also available through the `AttributePointer` class in `SysLib`. For more information of the corresponding methods in `SysLib`, see `AttributePointer Class` in the CA Aion BRE online help.

The `Constraints` attribute is a special attribute. It holds the constraint to be applied to attributes of this type. Because it is used to specify a functional feature of the Aion language, the following restrictions are imposed on this attribute

- It has to be of type **string**.
- Its access type must be **protected**.
- This attribute cannot be assigned except by edit time changes to its **initial value** field.

These constrained data types cannot themselves be used to specify a constraint. They must be subtyped before they can be used to specify a constraint on an attribute.

More Information:

[Operation of Constraints](#) (see page 115)

[Specify a Constraint](#) (see page 113)

Access Types

You assign an *access type* property to methods, attributes and constants. Access type determines which methods in the application can access them. Aion BRE provides three access types, allowing different levels of protection for each element.

Access Type	Element can be accessed
Private	Only by a method belonging to the same class.

Access Type	Element can be accessed
Protected	By a method belonging to the same class or to any descendant class.
Public	By any method or function in the application.

The default access type is Protected. To change access type to Public or Private, select the check box on the Properties tab page for the attributes or methods.

Accessor Methods

An alternative to giving attributes a Public or Protected access type, is to make them Private and then create accessor methods. An accessor method is a public method that accesses private attributes in the same class as itself. There are two accessor methods: one that gets the current value, and one that sets the attribute to a new value.

Using accessor methods makes it harder for you to accidentally change an attribute.

Comparing Terms

If you are familiar with other kinds of programming, you may be wondering how CA Aion BRE fits in with what you already know. The following table maps the Aion object-oriented terms to the closest term in database programming, the C language, and the C++ language. Aion and C++ terminology is parallel, since they refer to the same concepts.

Aion BRE Term	Database term	C term	C++ term	Java term
Class	Table definition	Struct	Class	Class
Attribute	Field or column	Field of Struct	Data member	Field
Instance	Record or row	Variable of type Struct	Object	Object
Method	Stored Procedure	Function	Member function	Method
Class attribute	Non-repeating attribute	Static global variable	Static member variable	Static field
Constant	Constant	#define	#define; Constant	Static final field

Anatomy of a Class

A class consists of attributes and methods. These are discussed on the following pages.

Attributes

An **attribute** is a value holder that can store a single value or a list of values, such as: 17, "George Washington", or 3,5,7,9. Like any value holder, an attribute has a data type and can only store values of that type. When you define an attribute in an application, you can specify its initial value. You access an attribute by its name.

Attribute Data Types

You specify a data type when you create an attribute. Aion provides a number of predefined data types, such as *string* and *integer*. To indicate that the attribute holds a list of values, use the language construct *list of* (*list of string* or *list of integer*). The system-supplied library called SysLib contains the definitions of the data types.

Data Type	Description	Example
Binary	A binary value can be any arbitrary sequence up to 2 gigabytes. Binary values can be used to build complex structures to pass to external methods.	bitmap image sound video complex structure
Boolean	Holds a value of either true (1) or false (0). The results of comparisons (for example, "If A > B") are Boolean values.	TRUE FALSE
<i>Classname</i>	The name of the class whose instances can be values of the attribute.	
<i>Constrained Data Type</i>	The name of a (subtype of a) constrained data type that specifies restrictions on the range of values of the attribute.	
Date	Holds any date, including year, month, and day. Use a date data type to create a formatted representation of the date value. Date values can be entered in	June 21, 1998 06/21/1998 21/06/1998 1998/06/21 21-June-1998

Data Type	Description	Example
	various formats, as specified by the application.	
Integer	Holds a four-byte integer value between -2,147,483,648 and +2,147,483,648. Typically, an integer is represented as a decimal (base 10), octal (base 8), or hexadecimal (base 16) number that represents a value.	1205 +33691 -7898
Real	Holds an eight-byte floating-point value between +1.0E+306. Financial data typically requires a real data type.	98.6 -000.02
String	Holds up to 65,512 characters, inclusive. Text data requires a string data type. String values must be surrounded by quotation marks. To specify quotation marks within a string, use two sets of quotation marks.	"yellow" "Jane Doe" "(800) 555-1212" "Error: ""Not found"""
Time	Holds any time value, to the nearest second, for dates after January 1, 1970. Includes the year, month, day, hour, minute, and second. Time values can be entered in various formats, as specified by the application.	June 21, 1998 4:15pm 06/21/1998 4:15pm 21/06/1998 16:15 1998/06/21 16:15 21-Jun-1998 4:15pm
Pointer to <i>classname</i>	The name of the class whose instances can be referenced by the value of the attribute.	
Classpointer to <i>classname</i>	Holds a pointer to the named class or any class from which that class derives (directly or indirectly). In this way, classpointer allows for an extra level of indirection beyond using a simple pointer to the named class.	
Attributepointer	Holds a pointer to an instance attribute or a class attribute that is not a constant. Attributepointers allow you to refer to another attribute dynamically.	

Most of the preceding data types are like those in other programming languages. However, further definition is required in the case of the following four data types, *classname*, *constrained data type*, *pointer to classname*, *classpointer to classname* and *attributepointer*.

Classname versus Pointer to Classname

If the attribute is of type *classname*, the relationship between the classes is containment. If the attribute is of type *pointer to classname*, the relationship is association.

For containment, consider a dialog box with a radio button. The dialog box class has an attribute of type *RadioButton*. Look at the dialog box class in the Project Workspace and find the attribute. Its name may have the letters *rb* in it, although this notation is not required. You can recognize an attached instance because the attribute displays an empty value and the Type column says: *RadioButton*. Even though you cannot see a pointer name, a pointer to the attached runtime instance does indeed exist. CA Aion BRE uses it to make containment work properly.

For association, consider the Department and Employee classes mentioned earlier. Each Employee instance has an attribute of type *pointer to Department*. They all point to the same instance of Department. Aion does not automatically create or destroy the Department instance named Marketing every time an Employee instance is created or destroyed.

Remember that for attributes of type *Classname*, Aion BRE automatically takes care of creating and destroying the instance. For attributes of type *pointer to Classname* you must create the associated instance, create a pointer to it, and destroy the instance when you are done with it. Failure to attend to these matters can lead to a memory leak.

Constrained Data Type

Aion BRE allows the programmer to define subtypes of the *Constrained<DataTypes>* types. These subtypes hold constraints on what values can be assigned to an attribute. CA Aion BRE will automatically assignments made to an attribute at both edit time and runtime to insure that any assignment will not violate the declared constraint.

Classpointers

An attribute of the type *classpointer* holds a pointer to the specified class or to any of *classname's* subclasses. Only class attributes and class methods can be accessed with the class pointer.

For example, you sometimes need to access class attributes and class methods when no instances of the class yet exist. You might write the following, where *pQuery* is a class pointer to a *Query* class.

```
pQuery.Flush( )  
pQuery.Load( )
```

Attributepointers

You can use an attribute of the type *attributepointer* to refer to another attribute. Through an attribute pointer you can retrieve the properties of an attribute and retrieve and set the value of an attribute.

The *attributepointer* data type is often used when using the meta-programming capabilities of Aion BRE.

User-Defined Data Types

Any of the predefined Aion BRE data types can be customized through inheritance and specialization. From your application, subclass a data type defined in the system-supplied library, *SysLib*. If you have several data types you use frequently, such as military time or European currency formats, you may wish to create an application that contains nothing but these custom data types. You can then use it as an included library in applications that need these data types. You can define a custom data type, and then reuse it in any number of text windows.

To define a custom data type

1. Click Logic, New Datatype
2. Enter the new datatype's class name
3. Select a base class
4. Press OK.
5. Then, customize the characteristics of the new class

Class Attributes

So far we have mainly discussed instance attributes. When we do not use a modifier in front of *attribute*, we mean *instance attribute*. An instance attribute can hold a different value for each instance.

A *class attribute* describes a characteristic of the class as a whole. It holds a single data value for the class, rather than a different value for each instance in the class. Class attributes are often used for counters, since the total number of instances is the same for all instances of a class. Like an instance attribute, a class attribute has a data type.

You also use class attributes when a class has no instances. For instance, the SQL SELECT statement is assigned to a class attribute of a query class. The reason for this is simple: the SELECT statement has to be issued *before* any records have been retrieved or any instances created.

To create a class attribute

1. Create an attribute.
2. In the Properties tab, click the Class Attribute check box.

Methods

Like a procedure or a function, a *method* contains logic to perform algorithms and calculations. In the body of a method, you access and modify values of attributes and local variables and call other methods.

Like attributes, methods come in two varieties: instance and class. When we say *method* without a modifier, we mean *instance methods*. An instance method operates directly on the data in an instance.

You indicate to which instance a method is bound by using a fully qualified name that specifies the instance or class. At other times, you can call a method without an instance or class name.

More Information:

[Call Methods of the Same Class](#) (see page 50)

Pass Arguments To Methods

Aion methods take two kinds of arguments, which differ in how they are passed to a method. An *input argument* is passed by value. An *output argument* is passed by reference.

Remember as well that a method can return a single value, which is considered the data type or return type of the method.

Input Arguments

Input arguments are passed by value. A copy of the input arguments value is passed to the method, and the method cannot change the value.

Note: An input argument can have a value of NULL.

Output Arguments

Output arguments are passed by reference. Output arguments must be variables, and a pointer to an output arguments variable is passed to the method. The method can then access the variable's current value and change the variable's value.

Specify Input and Output Argument

Output arguments are mandatory. If they are declared for the method, they must be specified in the method call.

Input arguments preceding output arguments are mandatory. Input arguments following output arguments are optional, if they are declared with a default value. If input arguments following output arguments are not declared with a default value, they are mandatory.

Local Variables

A method may have variables declared within the method body, which are created when control enters the method and destroyed when control exits. Hence, they cannot be accessed outside the method. You can use local variables for calculations and looping. A local variable is not an attribute of an instance or class attribute.

Class Methods

A counterpart to class attributes, class methods are also associated with the class itself, rather than a particular instance. Class methods exist and are callable before any instances of a class are created. Class methods access class attributes, or operate on groups of instances. SysLib provides the following frequently used class methods:

- `Create()` creates an instance and returns a pointer to the newly created instance. Once an instance is created, the application can begin invoking instance methods of the instance.
- `MessageBox()` uses a modal dialog box to display an informational or error message.

External Methods

In general, methods are considered to be internal, which means their implementation is written within CA Aion BRE using the Aion BRE language. However, you can also call external methods. An external method is implemented outside CA Aion BRE. Its implementation, or method body, exists outside CA Aion BRE as a named entry point in a dynamic link library (DLL).

Disabled Methods

In general, all methods are active. The disabled method is used primarily for system-invoked methods, or *events*. Because the method is defined, you can see its name and parameters. Because the method is not callable, however, there is no performance penalty when the program is executed. To define behavior for the method, specialize it within your application.

Under special circumstances, you might wish to create your own disabled methods.

Constants

A **constant** is a value holder whose value does not change during the execution of your program. It is a numeric or string literal that you want to refer to by name in your application.

In Aion a constant is a special kind of attribute, and shares the properties of an attribute:

- It is defined in a class.
- It has a data type.
- It has an access type.
- It can be inherited (though not specialized).

A constant is implemented as a class attribute whose value cannot change during execution.

To create a constant

1. Create an attribute in a class
2. From the attribute's Properties tab select the Class Attribute check box
3. From the attribute's Properties tab select the Constant check box.

Note: Prior to Aion 8.1.1, the CHAR_LF attribute for the _Datatype class in SysLib was defined as a constant. Since 8.1.1, CHAR_LF is defined as a non-constant. This change allows Aion to initialize CHAR_LF at runtime based on the operating system. As a result, applications that try to define their own constants and initialize them to a value that includes CHAR_LF will not parse or resolve at edit time. The solution is to initialize attributes that refer to CHAR_LF at runtime, preferably in _TaskInitialize() of the entry class.

Examples of constants:

Note: Constants are generally written in capital letters:

Constant	Type	Value
MAX_LENGTH	integer	10
INTEREST_RATE	real	7.85
RGB_COLORS	list of string	("red", "green", "blue")

Dynamic versus Static Instances

You work with instances continually while developing an Aion BRE application. You specify dialog boxes and windows for your application, insert icons and bitmaps in them, and specify SQL queries to retrieve data for analysis. These processes all involve instances of classes, yet they are not all the same kind of instance.

Static Instances

Also known as edit-time instances, they are the less frequent kind of instance in Aion BRE. You create a static instance at edit time, giving it a name by which you can access its attributes. Pointers are not necessary. When an Aion BRE application starts executing, the exact amount of storage is reserved for the static instances. This is possible because you know at edit time exactly how many instances will exist at runtime

In CA Aion BRE, you use static instances mainly in two circumstances:

- For the Connection instance that establishes a connection from an Aion BRE application to an external database.
- For the resources of an application-in particular, the bitmaps, cursors, and icons used in the GUI.

You can see static instances in the Aion BRE IDE. In the Project Workspace and the Aion BRE Explorer, static instances are represented by their own icon. You can right-click and view their properties. You can edit their properties in the Instance editor.

Dynamic Instances

These are also known as *runtime* instances. Most instances in Aion BRE are dynamic instances. You cannot tell at edit time how many dynamic instances of a class will be created when the program is executed. Think of a query that can result in an unpredictable number of records, each of which is an instance. Or consider a Multiple Document Interface (MDI) application, which allows any number of document window instances to be created and opened.

Since you cannot know how many instances of a class will exist at runtime, you do not name a dynamic instance. You need a pointer to the instance instead of a name. The new instances are allocated storage in the section of application memory called the free store, or the *heap*. Allocation to the heap grows and shrinks during program execution as dynamic instances are created and deleted. It is necessary to use the heap because an Aion BRE application cannot know at startup time how much memory to set aside for instances.

To access attributes of the instance, you must use the pointer with dot notation, as in the following:

```
pMyDialog.AttributeName.
```

If the class is a container, as MyDialog is, you can specify the attribute of an attached push button like this:

```
pMyDialog.MyPB.AttributeName
```

In the example, MyPB is of type PushButton. Aion keeps track of the actual instance for you and accesses its methods correctly.

Note: It is important to understand that dynamic instances do not appear in the Project Workspace or the Aion BRE Explorer. Since the instances only exist at runtime, there is nothing to display at edit time.

Implement Interfaces

Aion Interface Inheritance permits Classes to be associated with behaviors known as Interfaces. You can use interfaces as a means for defining a behavioral protocol that can be shared by otherwise unrelated classes.

A Class may directly implement Interfaces or it may inherit Interface implementations from ascendant Classes within the `_Object` hierarchy. Likewise, if a class implements an interface, it acquires not only the behaviors associated with that interface, but also the behaviors of interfaces ascendant to that interface within the `_Interface` hierarchy.

Develop Interfaces

All interfaces are directly or indirectly derived from a method-less class `_Interface`, `_Interface` is derived from `_System`. Interface development in Aion must conform to specific restrictions and standards.

Interfaces can declare *only* abstract public instance methods, they *cannot* declare:

- Class methods
- Private methods
- Protected methods
- Any form of data
- Implementations for any of its methods (except as noted on the following page)
- Variables
 - variables can not be defined to be of type `_Interface`
 - A variable can only be defined as a pointer to an `_Interface`

Interface development must include the following standards:

- An interface must be uniquely named. The interface can not share its name with another interface or class.
- There can be only one parent interface. The parent interface must be another interface.
- When inheriting methods, an interface inherits methods from its direct parent and all indirect parents.
- Parent and child methods, of the same name, must share the same signature:
 - Number of parameters
 - Ordered parameter types
 - Return value types

To create a new interface

1. From the Aion Logic menu, choose New, Interface.

The New Interface dialog appears prompting you to enter a name for your interface. `_Interface` is selected as the default parent interface in the base class field.

2. Enter the name of your new interface. Click OK.

A class editor opens with the new interface active.

3. Create the interface behavioral protocol by declaring methods to be associated with the interface.

Note: If the method has a return value, the method's implementation must specify a "Dummy" return statement, otherwise, no statements should be specified for the method's implementation. See following for sample dummy values.

```
return 0           //for an integer return value  
or  
return NULL       // for a string or pointer return value
```

Associate Interfaces to Classes

Although classes continue to be restricted to single inheritance, a given class can be associated with multiple interfaces.

To associate interfaces with a class

1. Double-click the class that you want to work with.

The class editor opens.

2. Select the Properties sheet tab.

3. In the Interfaces field name the interface(s) associated with the class.

Note: If the class implements multiple interfaces you must separate the interfaces with commas.

Example:

Class implementing multiple interfaces, given the following interfaces:

Interface Intf1 specifies two methods:

f() with no arguments & an integer return value

g() with no arguments & an integer return value

Interface Intf2 specifies two methods:

f() with no arguments & an integer return value

h(in p1 is integer) with no return value

If ClassA implements both Intf1 & Intf2, it must define or inherit public instance methods corresponding to all methods associated with each of its interfaces.

If any of the interfaces is, in turn, derived from other interfaces, ClassA must additionally define or inherit public instance methods corresponding to all methods associated with the ascendant interfaces.

Example:

In the preceding example, classA must define or inherit the following public methods:

f() with no arguments & an integer return value

g() with no arguments & an integer return value

h(in p1 is integer) with no return value

Note: The class implementation of the methods must have the same argument types and return-value type as in the interface declaration; otherwise, Aion flags the conflict as an error.

Associate Interfaces to Class Instances and Generic Methods

Once classes and interfaces are associated, class instances can be referenced via interface pointers.

Given the following interface:

Interface Flyable specifies three methods

Takeoff() with no arguments & a Boolean return value

Set Altitude(in altitude is integer) with no return value

Land(in destination is string) with a Boolean return value

With the following classes, Airplane and Duck-that implement that interface-you can write a generic method that applies to all classes implementing the interface:

```
FlyTo(in pObj is &Flyable,  
      in altitude is integer,  
      in destination is string)  
returns a Boolean value
```

Implementation:

```
var bSuccess is Boolean = false // be pessemistic  
if pObj.Takeoff( ) then  
    pObj.SetAltitude(altitude)  
    if pObj.Land(destination) then  
        bSuccess = TRUE  
    end  
end  
return bSuccess
```

and, in turn, invoke the generic method via instance and interface pointers:

```
// fly an airplane via an interface pointer  
var pFlyObj is &Flyable  
pFlyObj = Airplane.Create( )  
FlyTo(pFlyObj, 8000, "Phoenix")  
  
// fly another airplane via an instance pointer  
var pAirplane is &Airplane  
pAirplane = Airplane.create( )  
FlyTo(pAirplane, 5000, "New York")  
  
// fly a Duck  
FlyTo(Duck.Create( ), 50, "Redwood City")
```

Implicit Typecasting

The preceding examples illustrate Implicit Typecasting - The automatic conversion between instance pointers and interface pointers.

Typecasting occurs in assignment statements and parameter passing.

Cast Instance Pointers to Interface Pointers

Aion performs casting if the instance pointer's class (for example, Airplane) implements the interface pointer's interface; otherwise, the Aion Interpreter flags the usage as invalid.

Examples:

```
// assume airplanes are flyable
var pFlyObj is &Flyable
pFlyObj = Airplane.Create( )

// assume ducks implement some interface (any interface)
var pIntf is &_Interface
pIntf = pDuck
```

Cast Interface Pointers to Class Instance Pointers

Given the following interface:

```
interface Sortable specifies one method:
    Compare(in pOther is &Sortable) with an integer return value
```

In this case, the interface-method's pointer is a pointer to another Sortable instance. Classes that implement this interface must expect to convert such pointers to instance pointers

Example of a Duck method:

```
// Assume ducks are sortable & are to be compared.
// Return -1 if current duck weighs less than other duck;
// Return 1 if current duck weighs more than other duck;
// Return 0 if ducks weigh the same.
Compare(in pOther is &Sortable)
returns an integer value
Implementation:
var pOtherDuck is &Duck
pOtherDuck = pOther // typecast
if current.weight < pOtherDuck.weight then
    return -1
end
if current.weight > pOtherDuck.weight then
    return 1
end
// same-size ducks
return 0
```

Aion performs casting if the instance pointer's class (for example, Duck) implements that interface pointer's interface; otherwise, the Aion interpreter flags the usage as invalid

Cast Between Interface Pointers

If the pointers are associated with different interfaces, CA Aion BRE performs casting if one of the interfaces is derived from the other interface; otherwise, the Aion BRE parser flags the usage as invalid.

Example:

```
// assume that FlyableAnimal is an interface
// derived from interface: Flyable
var pFAnimal is &FlyableAnimal
var pFlyable is &Flyable
pFAnimal = ...
pFlyable = pFAnimal
var pIntf is &_Interface
pIntf = pFAnimal
```

No casting is required if the pointers are associated with the same interface.

Associated SysLib Methods

The SysLib.ClassPointer class defines the following methods that can be useful when working with interfaces:

GetInterfaces() identifies the interfaces implemented (directly or indirectly) by a given class.

GetInterfaceClasses() identifies the classes implementing (directly or indirectly) a given interface.

Inference Considerations

Pattern-matching bind-variables may be bound to interfaces (as well as to classes).

Example:

For a comprehensive example of Interface implementation see the \examples folder.

```
examples\intf_inh\intf_inh.app
```

Note: For additional information on Pattern Matching with bound interfaces, see the “Pattern Matching over Interfaces” chapter in the *CA Aion BRE Rules Guide*.

Chapter 4: How You Create and Edit Applications

This chapter explains how to create, edit, and manage applications and libraries using the CA Aion BRE development system.

Note: This chapter uses the terms *application* and *library* interchangeably.

This section contains the following topics:

- [Create Applications](#) (see page 79)
- [Save Applications](#) (see page 81)
- [Back Up Applications](#) (see page 81)
- [Restore Applications](#) (see page 82)
- [Develop Applications for Non-Windows Platforms](#) (see page 84)
- [The Command Line](#) (see page 88)
- [Customize the Development Environment](#) (see page 89)
- [View Applications](#) (see page 91)
- [The Output Window](#) (see page 93)
- [The Project Workspace](#) (see page 96)
- [The Explorer](#) (see page 99)
- [The Rule Analyzer](#) (see page 102)
- [Editors](#) (see page 102)
- [Create a Constrained Attribute](#) (see page 112)
- [Copy and Paste Objects](#) (see page 117)
- [Delete Objects](#) (see page 118)
- [Edit Toolbars](#) (see page 118)
- [Customize Toolbars](#) (see page 122)
- [Work with Included Libraries](#) (see page 122)
- [Work with Source Control](#) (see page 126)
- [Change Management](#) (see page 130)
- [Produce Reports for an Application](#) (see page 133)
- [Search for Objects by Name](#) (see page 136)
- [Search for Objects by Multiple Criteria](#) (see page 136)
- [Replace Text](#) (see page 136)
- [Search Across Applications](#) (see page 137)

Create Applications

For step-by-step instructions to create an application or library, see the section “Create an Application” in the CA Aion BRE online help.

.APP and .BIN Files

The .app file, called a *source* file, serves as a blueprint for the .bin-or *working*-files. The working files comprise not only the working copy of your application, but also files needed to debug, run, and build your application. You write to the .app file and the working files only when you choose Save from the File menu.

If the .bin files are out-of-date or missing when you try to open an application, CA Aion BRE prompts you to restore them from the application's source file. For example, if you use a source code management system, your local copy of an application's .bin files will be out-of-date when you "get" a new version of the .app file from the source code system after another developer has worked on that file. To update the .bin files, you must restore the application from its source, the newly "gotten" .app file.

More Information:

[Restore Applications](#) (see page 82)

Set Mainframe Line Length

When you transfer Aion BRE applications from the PC to the Mainframe, you can specify the line length and activate Line Wrapping to accommodate line length restrictions on the Mainframe host.

Use the Mainframe Deployment Options dialog to activate line wrapping and to set the line length that is allowed while saving the application file (*.app file)

For more information on setting the mainframe line length, see Setting the Mainframe Line Length in the CA Aion BRE online help.

Open Applications

For more information on opening applications, see Open an Application in the CA Aion BRE online help.

Note: Only one application can be open at a time in the Aion IDE. To edit multiple applications at the same time, open multiple sessions of the IDE. You can then cut and paste objects from one application to another across the various IDEs.

Save Applications

For step-by-step procedures on saving applications, see [Save an Application](#) in the CA Aion BRE online help.

Read-Only Applications

Read-only applications can be viewed and printed, but they cannot be modified. For more information on working with read-only applications, see the following topics in the CA Aion BRE online help:

- [Open a Read-Only Application](#)
- [Save a Read-Only Application](#)

Back Up Applications

For step-by-step procedures on backing up applications, see [Back Up an Application](#) in the CA Aion BRE online help.

Whenever you save an application, CA Aion BRE copies the old .app file to an `ApplicationName.bak` file located in the application's .bin directory.

Restore Applications

When you save an application, CA Aion BRE saves not only to the application's working files, but also to the application's source file, *ApplicationName.app*. Because the source file serves as a blueprint for the working files, you can regenerate-or *restore*-all the application's working files from this .app file.

The ability to restore an entire application from a single file provides the following benefits:

- **Easy backup and transfer**-You can save or move all of an application's working files by copying only the .app file.
- **Protection against unexpected events**-If the working files become corrupt, you can restore the application to the state it was in when last saved.
- **Quick library updates**-If an included library is modified, you can incorporate these modifications into the application simply by restoring the application from its source file.
- **Simple source control**-Since you can regenerate an application's working files from its source file, you can safeguard an entire application by checking only its source file in and out of a third-party source-code control program using the built-in support for SCC API-supported source control systems.

Note: If your application uses graphic resources (bitmaps, cursors, and icons) and ActiveX controls, you must also backup and transfer the files that contain these objects (namely .bmp, .cur, .ico, and ApplicationName.stg files) to restore the application successfully.

If these files are stored in any location other than an application's current directory, you may need to use the Instance editor's Properties tab page to modify the resources' paths before a transferred application's images can be restored.

Note: The case of the application file name under UNIX must be exactly the same as the case of the application name under the bound library4 sections, and inside any include file statements.

More Information:

[Create Resources](#) (see page 177)

Restore Closed Applications

There are several situations in which you must restore an application's working (.bin) files from its source (.app) file before you can open the application:

- When you check an .app file out from a source control program
- When you move an .app file from one machine to another
- When the application's working files are either out-of-date or missing

For step-by-step procedures on restoring closed applications, see Restore a Closed Application in the CA Aion BRE online help.

Open Before Restoring

If an application's .bin files are missing and you try to open the application's .app file before restoring the application, you receive the following message:

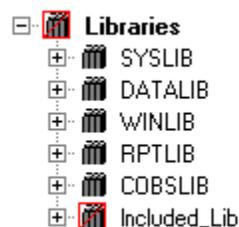
ApplicationName.app cannot be opened. Would you like to restore from source?

To restore and open the application, click Yes.

Restore Open Applications

When you modify an application's included libraries, you must update the application to incorporate the library changes into the application. To update an application, restore the application from its source file. Doing this incorporates all changes made to the included libraries since the application was last restored. You should restore an open application whenever you revise any of its included libraries.

If you do not restore an application after you update an included library, the time-and-date stamps of the application and of the included library become unsynchronized. When this occurs, CA Aion BRE places a red box with a slash through it over the Libraries node icon and over the icon of the out-of-sync included library (see the Libraries tab page of the Project Workspace):



Restoring the application from source resolves the discrepancy and removes the red boxes.

Note: When you add an included library to an application, that library's included libraries are also added to the application in piggyback fashion—that is, they are carried in by the directly included library. Like directly included libraries, piggybacking libraries can get out-of-sync with the main application. To bring such libraries into sync with an application, see the CA Aion BRE online help.

For step-by-step procedures on restoring open applications, see Restore an Open Application in the CA Aion BRE online help.

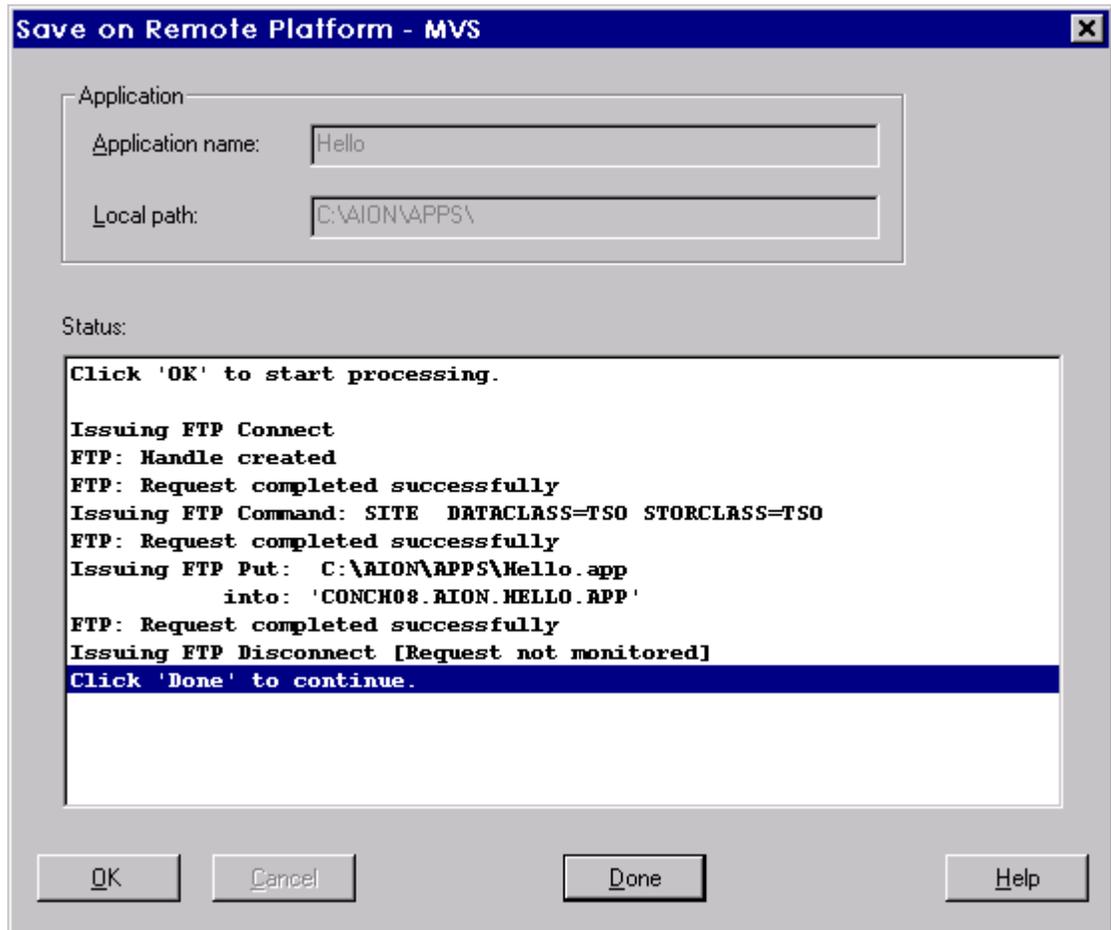
Develop Applications for Non-Windows Platforms

The Aion IDE is the standard environment for developing and editing of applications for mainframe (OS/390, z/OS, and Linux) and UNIX platforms (AIX, Solaris, HP-UX). For example, applications stored on the mainframe can be edited and run from the Windows environment.

Note: Using this development approach, only Aion BRE application (.app) files are transported between platforms, not binary (.bin) files or .dll files.

To access Aion BRE remote development capabilities, select Remote Development from the File menu. In order to use remote development, the Enable Remote Development check box on the Remote Development menu must be checked. You must also establish the settings for file transfer before saving to or restoring from a remote source. Select Remote Settings from the Remote Development submenu. The Settings dialog saves information to the Windows registry. You may establish settings to both the mainframe and a UNIX platform, because information of each of these connections is saved separately. For more information on establishing the settings for a remote connection, see Remote Settings in the CA Aion BRE online help.

All remote functions utilize dialogs similar to the following:



The title bar distinguishes which function is being performed and identifies the remote platform. The local file name is always displayed at the top, and below it a large Status list box. Functional pushbuttons are at the bottom.

- FTP processing is asynchronous; that is, an FTP command is issued, and then monitored until done; if successful, and more is to be done, then the next FTP command is issued.
- Processing may be interrupted by clicking Cancel. For Restore and Save, the transfer will be aborted. For Build, Run, and Submit, if the job has yet to be submitted, it won't be. If it has been, the job will in fact run to completion, but the output will not be fetched back for review. In all cases the dialog may be closed and other Aion tasks undertaken.
- Once processing is complete (or cancelled), the dialog remains open: click Done to close it and proceed with other Aion tasks. Or, click OK to perform processing again.
- The Status listbox is initialized with just the first line, "Click OK to start processing." Once OK is clicked, two sorts of messages are displayed: informational messages describing where Aion is in the process, and status messages from FTP reporting the status of the current FTP request. FTP messages are prefixed with FTP.
- All messages may be ignored for jobs that succeed. For debugging purposes, they are useful, indicating how far processing went, and all the file names used.
- The FTP Disconnect request is always last, and never monitored.

For step-by-step procedures for using the remote development capabilities of CA Aion BRE, see the following topics in the CA Aion BRE online help.

Note: These options are not available if Remote Development is not enabled. If the remote platform is UNIX (based on the Settings), Build, Run, and Submit are grayed out. If no application is open, Save, Build, and Run are grayed out. If an application is "active" (open, changed, but not yet saved), Restore is grayed out.

- Restore Applications from Remote System

The Restore Remote operation invokes the Aion internal FTP client (reftp) to download a copy of an MVS or UNIX application to the Windows platform. The application is restored from source, and the .bin file is built locally in Windows. Local development can then occur as with any other Windows application. The local copy of the application can be saved with File, Save. If desired, it may also be built locally for local testing.

Note: If no application is currently open, you can specify any application, by name, during Remote Restore. But when there is an application already open, then only that application may be remotely restored. An application must be unchanged and saved for it to be restored.

- Save Applications to Remote System

The Save Remote operation saves the application locally. It then invokes the Aion internal FTP client (reftp) to upload a copy of the application to the host platform. Any file may be saved remotely, whether or not it was restored remotely. Applications may also be restored remotely for local use. For existing applications, the old MVS or UNIX dataset is overwritten.

Note: Any application can be remotely saved, whether or not it was remotely restored. The application must be currently opened.

- Build Applications on a Remote Source (MVS only)

Remote Build builds the remote version of an application on the mainframe. The results of the build are returned back to the Windows environment for review. The first step is to restore the application that is already resident on the remote platform.

Note: Remote Build uses the version that is currently available on the remote platform. For this reason, invoking Remote Build is preceded by a prompt for a Remote Save.

- Run Applications on a Remote Source (MVS only)

Remote Run invokes program REEXEC on the mainframe and runs the remote copy of an application interpretively. Results are received back for user review.

Note: The current remote version of the .bin file is used, as last created (for example, by Remote Build. For this reason, invoking Remote Run is preceded by a prompt for a Remote Build.

- Submit Jobs to a Remote Source (MVS only)

Remote Submit allows the application developer to submit any JCL file to the mainframe. This permits the application developer to execute any mainframe program (for example, a built Aion BRE application, DB2 bind job, COBOL compiles, or clean-up jobs) from the Aion BRE development environment running under Windows. Another use for Remote Submit is to accommodate special Remote Build or Remote Run needs. For example, if an application writes to a SYSOUT file not defined in proc BARUN, the JCL generated by Remote Run, *localpath.appname.run*, may be edited to add the additional DD card. This same Run dataset can be specified as input to Remote Submit.

Results of a remotely submitted job are returned to the Windows environment.

Note: There is no automatic clean up of work files and FTP submitted jobs under MVS. It is the user's responsibility to effect proper clean-up of the remote platform. Ideally, work files should by default be written to work disks and be automatically deleted. Alternatively, steps may be added to the procs to perform deletes, or, JCL may be written to invoke the standard BADELETE procs and submitted using Remote Submit. For additional information on cleaning up the job queue, see Clean-Up Procedures in the CA Aion BRE online help.

The Command Line

CA Aion BRE lets you work with applications from both the Windows and UNIX command lines. When working with applications from the command line, you must include the complete path name of the application, unless it is in the same location as the Redev Aion executable. Following are the available line commands and their options:

Command	Options	Explanation
reexec	-	reexec c:\aionnn\myapp.app Runs the application.
	debug	reexec c:\aionnn\myapp.app -redebug Displays the application in the Aion debugger.
respawn	-	respawn c:\aionnn\myapp.app Opens the application and forces a restore from source if the application cannot open.
	clean	respawn c:\aionnn\myapp.app -clean Removes generated code, object files, and more.

Command	Options	Explanation
	comp	respawn c:\aionnn\myapp.app -comp Compiles generated code.
	gen	respawn c:\aionnn\myapp.app -gen Generates code for application.
	link	respawn c:\aionnn\myapp.app -link Creates shared library.
	res	respawn c:\aionnn\myapp.app -res Restores application only.
	trace	respawn c:\aionnn\myapp.app -trace Enables tracing in generated code.
redev		redev c:\aionnn\myapp.app Opens the application. If no application is specified, it opens redev.

Customize the Development Environment

Use the Settings dialog to set directory, run, and build defaults for your Aion development environment.

To open the Settings dialog, choose Settings from the File menu.

Note: Changes made in the Settings dialog take effect immediately. You do not need to exit and restart Aion to apply them.

Directories Tab Page

In the Directories tab page, you specify system locations for accessed and generated files.

- In the Included Libraries Path field, you can specify the full path for any library (whether custom-built or supplied by Aion) that you include in an application. This field can contain paths to multiple directories. Place a semicolon (;) between each path as follows:

```
C:\Program Files\CA\AionBRE\init;
```

During a build, if your application includes a custom library, Aion looks in the directories specified here for the library's LIB file.

Note: The information contained in this field can also be modified in the System field of the Included Library Editor. In addition, you can use the Included Library Editor's Application field to specify paths to custom libraries included in your application.

For more information about the System and Application fields, see the Included Library Editor Fields in the [Include and Remove Libraries](#) (see page 123).

- In the Working Directory field, you can specify a default directory for file operations (from the File menu, choose New, Open, Restore from Source, Delete, and Save As). The specified directory is preloaded into the Create In, Look In, Delete From, or Save In fields of the dialogs that open when you select these operations. By specifying a commonly used directory here, you may save time when using the File menu to access or generate files.

If you do not specify a directory in the Working Directory field, the current directory is loaded into the file operation dialogs.

- In the Trace Output Directory field, you can specify the directory in which the Execution Trace Output file (*ApplicationName.trc*) is stored. If a location is not specified here, the Trace file is written to the directory designated for temporary files by Windows.
- The AionDoc Directory field specifies where AionDoc HTML files are saved.
- Verify that the Build Directory field lists the Aion Build directory:

```
<INSTALL_DIR>\build
```

This is the directory that CA Aion BRE uses to locate the system files it needs to build an application. The default value in this field is automatically set during installation and should not require modification. If no directory is specified in this field, you cannot build your application.

- In the Compiler Directory field, you must specify the top-level directory where the compiler is stored. For example C:\Program Files\Microsoft Visual Studio 8.
- The Java Home Directory box specifies the root path. For example C:\Program Files\Java\jdk1.6.0_03 The the following directories are in the Java Home Directory box:
 - %java_home%\bin\javac
 - %java_home%\bin\jar
 - %java_home%\include
 - %java_home%\include\win32

More Information:

[Run and Build Applications](#) (see page 439)

View Applications

CA Aion BRE displays information about an application in a variety of windows:

- Output Window
- Project Workspace
- Explorer
- Editors

These windows open inside the Aion main frame window.

Manipulate Windows

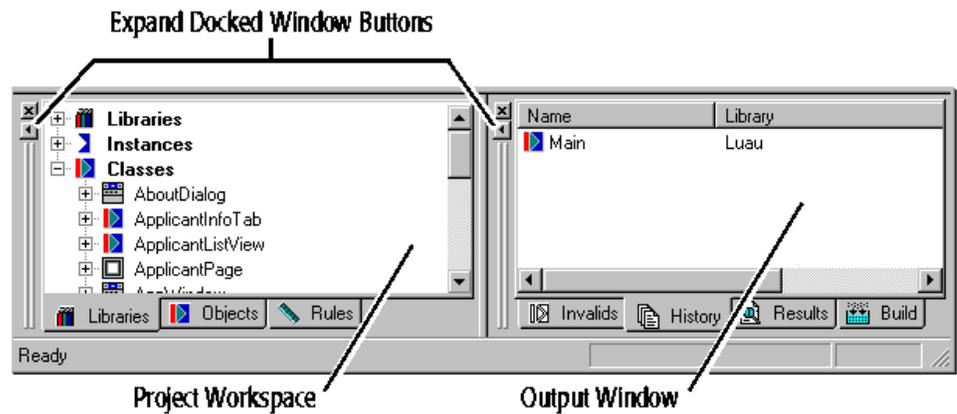
You can size the Aion main frame window and the windows it contains.

For step-by-step procedures on manipulating windows in the following ways, see Manipulating Windows in the CA Aion BRE online help:

- Resizing
- Detaching a dockable window
- Redocking
- Disabling and re-enabling the docking feature

Enlarge the Main Frame Work Area

If you move the Project Workspace window on top of the Output Window when the Output Window is docked at the bottom of the main frame window, Aion will display the two windows side-by-side in the space formerly occupied by the Output Window alone. This arrangement enlarges the remaining work area of the main frame window, giving you more room to display the editors. You can expand either window across the other by clicking the Expand Docked Window button (located near the top left corner of each window as shown in the following figure). To redisplay both windows, click the button again.



Object Icons

Each object name displayed in the Aion IDE is preceded by an icon that indicates the object's type. The Aion IDE icons are:

	Class		Attribute		Method		Instance
	Library		Class Attribute		Class Method		Rule
	Window		Query		Menu		Toolbar
	Dialog box Box		Stored Procedure		Menu Item		Tool Item



Menu Bar

CA Aion BRE has many toolbars, but only one menu bar. When you open an application, Aion displays nine options in the menu bar:



The availability of these options and of the items on their sub-menus depends upon your location in the IDE. Whenever the following sections of this chapter refer to a "menu bar option," they are referring to one of the options shown in the preceding figure.

Fonts

For step-by-step procedures on modifying font characteristics, see [Modifying Fonts](#) in the CA Aion BRE online help:

You can modify font characteristics for the following types of text in the IDE:

- Imagelist
- Treelist
- Method Editor
- Decision Table cells

The Output Window

The Output Window displays information about operations performed during the development of an application.

When you first open CA Aion BRE, the Output Window is the only window displayed. If you exit CA Aion BRE when the Output Window is closed, no window will be displayed the next time you open Aion.

For step-by-step procedures for opening or closing the Output Window, see [Using the Output Window](#) in the CA Aion BRE online help.

More Information:

[Output Window Tab Pages](#) (see page 94)

Output Window Tab Pages

The Output Window contains four tab pages: Invalids, History, Results, and Build. To display a tab page, click its corresponding tab at the bottom of the window.

Display Headers

To display column headers on any Output Window tab page that contains columns, click the Output Window, and then from the View menu, choose Show Labels.

To adjust the width of columns, size the headers with your cursor.

Open Editors

To open the editor for any object displayed on a tab page, double-click the object.

More Information:

[Editors](#) (see page 102)

Invalids

The Invalids tab page lists application objects that contain errors, the name of the library in which these objects reside, and a description of each error. Whenever objects are listed on the Invalids tab page, a red box with a slash through it appears on the Invalids tab and in the status bar.

History

The History tab page maintains a running list of the objects you open during an Aion BRE session. This list speeds the development process by making it easy to return to the same objects again and again. To reopen objects in their editors or to move open objects to the top of the main frame window, simply double-click their name on this list.

Objects are displayed on the History tab page as follows:

- When an object is first opened, its name is added to the top of the list.
- When an object is closed, its name remains on the list, moving lower as subsequent objects are opened.
- When an object's editor is relayered in the main frame window, the object's name is repositioned in the list to reflect its new position among the open editors.
- When an object's editor moves to the top of the main frame window after another editor closes, the object's name moves to the top of the list.
- If no other objects are open when you close an object, the list remains unchanged.
- When an object displayed in the list is deleted from the application, the object remains in the list but is preceded by a grayed icon.

Such objects are inaccessible.

For step-by-step procedures for clearing the History and Results tab pages, see Output Window Tab Pages in the CA Aion BRE online help.

Results

The Results tab page displays the results of search and search-and-replace operations. Each top-level node in the display is a description of one operation. The objects listed beneath each top-level node are objects containing the name or string specified in the particular operation. To hide (or collapse) these subnodes, click the box containing a minus sign next to the search description.

The most recent search appears at the bottom of the list.

Build

The Build tab page displays the progress and results of the current Build operation (from the File menu, choose Build), including compiler and linker errors.

The Project Workspace

The Project Workspace window enables you to view an application's classes and included libraries from various perspectives. By default, the Project Workspace opens automatically when you open an application. There are, however, other ways to open and close this window.

For step-by-step procedures for other ways of opening and closing the Project Workspace, see Using the Project Workspace in the CA Aion BRE online help.

View Inheritance and Ownership Information

The Project Workspace displays various groups of classes. Within these groups, classes are arranged alphabetically rather than on the basis of inheritance. You can, however, view the class from which a class has been derived (known as a *base class*) in the Project Workspace as follows:

To display the name of a class's base class in the Project Workspace

- From the View menu, choose Show Parent.

You can also view the attributes and methods owned by a class:

To display objects owned by a class in the Project Workspace

- Click the plus (+) icon in front of the class's name.

More Information:

[The Explorer](#) (see page 99)

Project Workspace Tab Pages

The Project Workspace contains five tab pages: Libraries, Classes, Rules, Domain Interface, and Exports. To display a tab page, click its corresponding tab at the bottom of the window.

Open Editors

To open the editor for any object displayed on a tab page, double-click the object, *or* right-click the object and select Open from the pop-up menu.

Note: Double-clicking a *library* on the Libraries tab page will open the library in another session of CA Aion BRE. To open an included library's *class* in the current session of Aion, simply expand the appropriate library and then double-click the class.

More Information:

[Editors](#) (see page 102)

Libraries

The Libraries tab page alphabetically lists the application's included libraries, static instances, classes, and class members in a tree control that can be expanded and collapsed by clicking the minus and plus signs displayed on the tree's left. To display the base classes of all the classes listed on this tab page, highlight any object on the tab page, and from the View menu, choose Show Parent.

Expand the Libraries node to view a list of the libraries included in the application. Libraries can be included both directly (through the Included Library Editor) and indirectly (as included libraries of directly included libraries). To see which is which, choose Edit Library Includes from the File menu. The indirectly included libraries will not be listed in the Included Libraries field of the Included Library Editor dialog.

Expand the Instances node to view a list of the application's static instances. In Aion BRE applications, static instances are usually connections, resources (bitmaps, cursors, and icons), or user-created objects.

Expand the Classes node to view a list of the application's classes. To view the methods and attributes owned by a particular class, expand that class's node.

Classes

The Classes tab page facilitates object reuse by providing quick access to developer-created classes. The page divides these classes into seven groups:

- Windows and Dialogs
- Groups and Pages
- Menus and Tools
- Queries and Stored Procedures
- Associations
- _Interfaces
- User-Specified Classes

When you create a class that belongs to one of the first six groups, CA Aion BRE automatically adds that class to the appropriate group in the Classes tab page. User-Specified Classes, is designed to hold classes that cannot be categorized in any of the other groups. You must manually add classes to the User-Specified Classes group.

Note: Classes displayed on the Classes tab page come not only from the open application, but also from the application's included libraries. Classes supplied by Aion cannot be added to this page.

For step-by-step procedures for adding a class to the User Objects group, see Project Workspace Tab Pages in the CA Aion BRE online help.

Rules

The Rules tab page lists all the classes in the application that own methods containing rules.

For step-by-step procedures for displaying rules on the Rules tab page and opening the Method Editor for a specific rule, see Project Workspace Tab Pages in the CA Aion BRE online help.

More Information:

[Method Editor](#) (see page 107)

[Method Editor](#) (see page 190)

Domain Interface

The Domain Interface page lists all the classes in the application that own domain interface members. To display the domain interface members on the Domain Interface tab page, expand the classes listed on the Domain Interface tab page. A list of the domain interface labels that have been assigned to domain interface members (methods) of the class is displayed beneath each class.

To open the Method Editor for a specific domain interface label, double-click the domain interface label on the Domain Interface tab page. Click the Properties tab on the Method Editor and go to the Domain Interface Member Definition page on the Properties page, or click the Implementation tab on the Method Editor.

More Information:

[Method Editor](#) (see page 190)

[Method Editor](#) (see page 107)

Exports

The Exports tab page lists all the classes in the application that have been designated as Export classes in the Class Editor (Property page). The public methods of exported classes constitute the exposed methods in the interface layer of an Aion component.

For step-by-step procedures for displaying the public methods on the Exports tab page and opening the Method Editor for a specific method, see Project Workspace Tab Pages in the CA Aion BRE online help.

More Information:

[Class Editor](#) (see page 106)

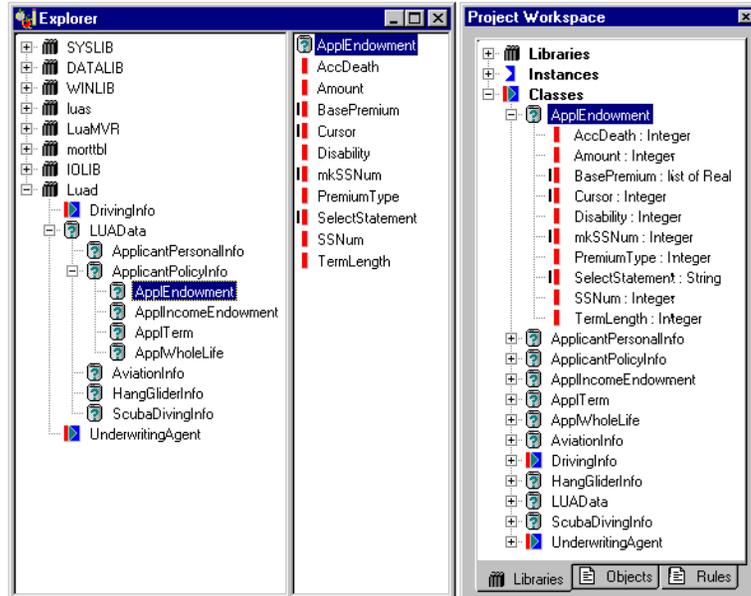
[Method Editor](#) (see page 107)

[Method Editor](#) (see page 190)

The Explorer

You can view the *genealogical structure* of an application's classes to see how a particular class is related to other classes by opening the Explorer. The Explorer depicts relationships among the application's classes based on the object-oriented concept of inheritance. In contrast, the Project Workspace provides an *alphabetized listing* of an application's classes. If you know the name of the class you are seeking but are unfamiliar with the class's lineage, the Project Workspace is the best place to look.

The Explorer contains two panes. In its left pane, it displays a class tree. In its right pane, it displays a list of objects owned by whichever class is highlighted in the left pane. To see how the views provided by the Explorer and the Project Workspace differ, compare the position of the ApplEndowment class in the Explorer (left panel) with its position in the following Project Workspace:



More Information:

[The Project Workspace](#) (see page 96)

Set Explorer/Workspace Options

You can use the Explorer/Workspace Options dialog to set options that allow you to expand a library and view your local classes and all classes found in libraries that include the expanded library.

From the Explorer/Workspace Options dialog you can set the Open Multiple Explorers option so that Aion will allow you to have multiple Explorer windows running in the active Aion session. You can also set the Domain Interface Group By Libraries option, which causes the Domain Interface tab page to show domain interfaces across libraries rather than grouping them by class.

For step-by-step procedures for customizing the Explorer, see Setting Explorer/Workspace Options in the CA Aion BRE online help.

Specialize Through the Explorer

The Explorer shows each object's place in an application's hierarchy, enabling you to see both the classes from which the object inherits characteristics and the classes to which it passes its own unique characteristics. This scoping information makes it easy for you to see how widespread the effects of your specializing and unspecializing operations will be.

In addition, if you must specialize or unspecialize a lot of objects, it is more efficient to do so on an object-by-object basis down a branch of this hierarchy rather than in a more disjointed manner outside the inheritance context, such as in the Project Workspace.

Discover Where an Object Resides

The fastest way to find out where an object resides in an application is to execute the Explore command on the object.

Example:

1. Select a method in Winlib from the Project Workspace.
2. Open the Explorer. CA Aion BRE quickly drills down to the method's home.

View Options for the Explorer

To customize the Explorer's display, use the following options on the View menu:

- Select Show Inherited to display objects inherited from the base class of the selected class. By default, only objects defined in a class are listed in the Explorer's right pane.
- Select Show Publics Only to display only objects whose access type is public (versus private or protected).
- Select Show Parent to display either the base class (in the case of classes) or the owning class (in the case of class members) in brackets to the right of each object listed in the Explorer.
- Select Show Labels to display column headings in the Show Details view. This option does not take effect unless Show Details is turned on.
- Select Show Details to display each object's data type, access type, and owning library and to indicate whether the object is inherited or specialized.
- Select Expand Current to display the subclasses (if any) of a selected class.

To fully expand a hierarchy, select a class or library in the Explorer, and press Shift+Enter.

The Rule Analyzer

The Rule Analyzer displays the rules that can change the value of a given attribute. Use the Rule Analyzer to perform the following tasks:

- Determine the rules that influence an attribute.
- View the text of a rule or attribute that is selected.

For step-by-step procedures for opening the Rule Analyzer, see Using the Rule Analyzer in the CA Aion BRE online help.

From the Rule Analyzer, you can view the organization and relationship between rules and the attributes they use. You can also view these relationships by checking the Forward Mode option in the right click pop-up menu. When the Forward Mode option is not checked, the Analyzer is in Backward Mode.

- Use Forward Mode when you want to know which rules will fire when you assign a value to an attribute, and what attributes will be assigned values as a result.
- Use Backward copy Mode when you want to see which rules can assign values to an attribute, and to determine which attributes are needed to get a rule to fire.

Editors

CA Aion BRE has many editors. Most of the application development process takes place in these editors. This section describes tab pages, procedures, and then provides a brief overview of each following editors.

- Association Editor
- Class Editor
- Decision Table Editor
- Format Editor
- Instance Editor
- Menu Editor
- Method Editor
- Rule Editor
- Query Editor
- Stored Procedure Editor
- Tool Editor
- Window Editor

Standard Tab Pages

The following tab pages are common to many editors:

- The Properties tab page enables you to view and modify values for the open object.
- The Uses tab page displays objects to which the open object refers.
- The Used By tab page displays objects that refer to the open object.

Standard Procedures

You can perform the following functions in each of the editors:

- You can open application objects in the appropriate editor
- You can open the base class and subclasses of a class in an editor
- You can save work done in editors
- You can discard work done in editors

These procedures are described on the following pages. To learn how to perform procedures that are specific to a particular editor, see the cross-reference listed in that editor's description under Editors.

Editors are sizable, floating windows that open in the Aion main frame window. You cannot open an editor. Instead, you must "open" an object, thereby opening the particular editor in which the object is displayed and changed.

CA Aion BRE imposes no limit on how many editors can be open at the same time in a single session. Your memory and system resources, however, may impose limits.

When you open an editor, the Aion BRE menu bar changes to display the options available for that editor. Some of the options, including Save and Close on the File menu, apply to the *entire* application, not just to the open editor.

Important! Following the Save or Close Editor procedure adds your changes to the working application, but it does not add your changes to disk. You must choose Save from the File menu to save your changes to disk.

For step-by-step procedures for the listed operations, see the following topics in the CA Aion BRE online help:

- Opening Objects in Editors
- Opening Base Classes and Subclasses from Editors
- Saving Changes in Editors
- Discarding Changes in Editors

Association Editor

Use the Association Editor to perform the following tasks:

- Define associations between classes
- Specify the roles that each class plays in the association

By an association is meant a set of reciprocating attributes between two classes. For example, classes may be associated as owner and owned, husband-of and wife-of, or whole and part. In traditional programming, associations are typically implemented by pointer attributes: each class in the association maintains a pointer or list of pointers to (instances of) its associated class.

In CA Aion BRE, associations between classes are often handled by creating a subclass of SysLib's `_Association` class rather than by hard-coded pointers. For example, a subclass of `_Association` called `Ownership` might define a relationship between the class `PropertyOwner` and the class `Property` that governs how an application links various instances of these classes. The `_Association` class provides the methods and attributes necessary to define the structure of an association you wish to establish and to facilitate coding operations on that association.

The Association Editor makes it easy for you to create subclasses of the `_Association` class. The Association Editor allows you give a specific name to the association, for example, `Ownership`, or `Marriage`, and to define the following properties of the association:

- Multiplicity constraints on the number of instances of each class that are permitted to be associated within a single association
- The roles played by each class in the association. (Roles provide public-like attributes of the associated classes that allow programmatic navigation of the association.)

For step-by-step procedures for using the Association Editor, see Using the Association Editor in the CA Aion BRE online help.

For greater flexibility in your designs, the Association Editor permits you to define associations over `_Interfaces`. For example, aggregation could be defined as an association over `_Interfaces` `IWhole` and `IPart`. The advantage of using `_Interfaces` in this case is that you can now create a bill of materials structure of aggregates of aggregates using just one association. A class representing an element in the bill of materials could implement both the `IWhole` and `IPart` `_Interfaces`.

For more information about associations, see the following in the CA Aion BRE online help:

- Associations
- Using Associations
- Association Class

Attribute Editor

Use the Attribute Editor to perform the following tasks:

- Change attribute properties such as name, data type, access type, and initial value
- Record comments about the attribute
- View lists of objects that use and that are used by the attribute

More Information:

[Attributes](#) (see page 213)

Class Editor

Use the Class Editor to perform the following tasks:

- Change the name of a class
- Designate a class as the application's entry class
- Add a class to the Objects tab page under User Objects
- Export a class
- Specify interfaces for a class (see Develop Interfaces)
- Record comments about a class
- View lists of objects that use and that are used by a class
- View a list of the methods and attributes owned by a class
- Specialize inherited members of a class

Members Tab Page

For step-by-step procedures for using the Class Editor to specialize class members, see Using the Class Editor in the CA Aion BRE online help.

More Information:

[Set Explorer/Workspace Options](#) (see page 100)

[Object Generation Overview](#) (see page 390)

Decision Table Editor

A decision table consists of a set of conditions and a set of actions. It is similar to a collection of IFRULES with related premises and actions. Decision tables display logic and decision flow in a graphical, easy-to-follow format.

Use the Decision Table Editor to create and maintain decision tables. Creating a decision table includes opening the Decision Table Editor, adding new Conditions and Actions, and generating the table.

More Information:

Editing Decision Tables, see Creating and Opening Decision Tables in the "Decision Tables" chapter of the CA Aion BRE *Rules Guide*.

Instance Editor

Use the Instance Editor to perform the following tasks:

- Modify attributes (such as filename and handle) of static instances
- View a list of objects that use and that are used by the instance

More Information:

[Create Resources](#) (see page 177)

[Dynamic versus Static Instances](#) (see page 70)

Menu Editor

Use the Menu Editor to edit menus for graphical user interfaces. The Menu Editor displays a graphical representation of a menu title and its menu items. The position of each item in the Menu Editor's display indicates where it appears on the menu at runtime.

When the Menu Editor is open, you have access to the following tools:

- Commands on the Screen menu that you can use to:
 - Add titles and items to the open menu
 - Re-arrange the items in each title contained in the open menu
- Add Menu Item button on the Menu Create toolbar
- Properties dialog for each menu title and item
- Method Editor, in which you can create event-triggered methods such as WhenChosen and WhenSelected for menu items you create in the editor

More Information:

[Add Controls to Windows](#) (see page 151)

Method Editor

Use the Method Editor to perform the following tasks:

- Change method properties such as name, data type, access type, and style (class, external, disabled, event, or DI member)
- Record comments about the method
- View and modify the method's input/output arguments and its body

Implementation Tab Page

Use the Method Editor's Implementation tab page to edit a method's input/output arguments and its body. The Method Editor toolbar and the mouse pop-up menu can help you format and parse logic statements on this page. You can also edit arguments on the Properties page.

For step-by-step procedures for opening the editor for a method and for invoking help for a method, see *Using the Method Editor* in the CA Aion BRE online help.

Customize the Method Editor

You can modify the way code is displayed on the Method Editors Implementation page by using the Method Editor Options. The values set in this dialog are recorded in your system registry, not in the application's files, so they apply to all CA Aion BRE sessions run on your machine.

For step-by-step procedures for customizing the Method Editor's Implementation page, see *Customizing the Method Editor* in the CA Aion BRE online help.

More Information:

[Choose Fonts](#) (see page 144)

[Method Editor](#) (see page 190)

Rule Editor

The Rule Editor provides a structured environment for writing and editing rules. Use the Rule Editor to perform the following tasks:

- Browse the rules in an entire rule method.
- Easily add or remove rules by cutting or pasting items.
- Edit all the properties of a rule.
- Add new rules using the Logic menu options: New, If Rule, When Rule, When Match, or If Match.

For step-by-step procedures for opening the Rule Editor, see *Using the Rule Editor* in the CA Aion BRE online help.

Properties Tab Page

Use the Properties tab page to set the following properties for the Rule Method:

- Method Name
- Access type
 - Public
 - Protected
 - Private
- Comments
- Class Method (Use the check box to set the Class Method property)

Rules Tab Page

Use the Rules tab page to toggle the current rule method, and to edit the following properties for each rule:

- Rule Name
- Priority
- Demon Rule (use the check box to set the Demon Rule property)
- Comments
- Premise
- Action

More Information:

For Rules and Inferencing see the *CA Aion BRE Rules Guide* and CA Aion BRE Rules online help.

Query Editor

Use the Query Editor to perform the following tasks:

- Browse through a catalog of database tables and of the columns within those tables
- Define field attributes mapped to the table columns
- Change the query's Select statement

- Change the properties of a query. This facility controls what information is retrieved from the database by the query editor, what conventions are used to construct the SQL statement, how NULL database values are treated in Aion, and whether static or dynamic SQL will be used (dynamic SQL is the default).

Note: To change the properties of a query, it is necessary to access the Query Properties dialog:

1. Open the query editor for the query class that you want to work with.
2. In the top right pane of the Query Editor, right-click the query class icon.
3. Select Properties from the pop-up menu. The Query Properties dialog will open.

More Information:

[Query Editor](#) (see page 224)

Stored Procedure Editor

Use the Stored Procedure Editor to perform the following tasks:

- Browse through a catalog of a database's stored procedures and of the parameters within those stored procedures
- Define markers mapped to the parameters
- Change the stored procedure's Execute statement

More Information:

[Access Data](#) (see page 217)

Tool Editor

The Tool Editor displays a graphical representation of a toolbar. Use this editor to create toolbars for graphical user interfaces.

When the Tool Editor is open, the following features are available:

- Commands on the Screen menu that you can use to
- Add tools to the open toolbar
 - Rearrange the tools contained in the open toolbar
- The Add Toolbar Item button on the Menu Create toolbar
- The Properties dialog for each toolbar and tool item
- The Method Editor, in which you can write event-triggered methods such as WhenChosen and WhenSelected for tools

More Information:

[Add Toolbars to Windows](#) (see page 171)

Window Editor

Use the Window Editor to construct windows and dialog boxes.

When the Window Editor is open, you have access to the following tools:

- Color and font bars to modify a control's color and text
- Layout toolbar to organize the elements of the window or dialog box
- Properties dialog to change appearance, behavior, and title of the window or dialog box
- Screen menu to create controls and add event-triggered methods to them
- Test mode to preview the runtime appearance of the window or dialog box
- Window Editor toolbar to add controls and graphics to the window or dialog box

More Information:

[Work with Windows and Dialog Boxes](#) (see page 146)

Create a Constrained Attribute

CA Aion BRE allows the programmer to specify constraints on attributes, that is, to restrict the possible values of an attribute to a specific subset of the total values that an attribute of the given type may have. For example, a String attribute, which can have an infinite number of values, may be restricted to just the seven values that comprise the days of the week. In Aion, such constraints are provided through the mechanism of a *constrained data type*.

The procedure for defining a constrained data type is described in [Specifying a Constrained Data Type](#) in the CA Aion BRE online help.

The following section describes, in detail, how to construct the specification of a constraint—in other words, how to specify the Initial value of the Constraints attribute.

More Information:

[Overview of CA Aion BRE Objects](#) (see page 43)

Specify a Constraint

The basic constructions used to specify a constraint are:

Is from(*constraintExpression*) or Is not from(*constraintExpression*)

Where *constraintExpression* is an atomic expression or a combination of atomic expressions separated by a comma. Atomic expressions consists of the following expressions:

- The keyword NULL
- A specific integer, real, string, or Boolean value
- A Range expression (not supported for strings and Booleans)

Integer values may range between +/- 2147483646; real values may range between 1.0E+/-306. Strings must be quoted in constraint expressions.

Range expressions specify ranges of values and are formulated similarly to the syntax used in the decision table editor, that is, ranges between numerical limits are expressed with the ".." operator. The following constraint operators are available:

- =, <, <=, >, >=

Note: It is possible to define attribute constants in terms of Constrained Data Types. The constraint would be restricted to the single value that is the value of the constant attribute.

Note: Because constraints are specified as a string in the Initial Value field of the Constraints attribute, internal strings need to be double quoted; for example:

```
"is from("Green", "Red")"
```

Examples:

Valid range expressions are:

- <= 100
- > 0
- = 1 (equivalent to a constant attribute)
- >20.<50
- >=20.<50
- >20.<=50
- >=20.<=50

Valid values for constraints on integers are:

- Is from (NULL, >=1..<=100)
- Is from (>=20..<30, >=70)
- Is not from (-25, 10, 20, NULL), which is equivalent to
 - Is not from (=-25, =10, =20, =NULL)
 - Is from (35), which is equivalent to a constant attribute

Valid values for constraints on strings are:

- Is not from ("Gray", "Black")
- Is not from ("Black")
- Is from ("Red"), which is equivalent to a constant attribute
- Is from ("Red", "Blue", "Green", "Yellow", NULL)

Valid values for constraints on Booleans are:

- Is not from (NULL)
- Is from (TRUE, FALSE)

Invalid value ranges, such as the following, will be detected at edit time:

- Is from (>=20..>100)

Note: Redundant constraint expressions such as Is from (NULL, NULL) or Is from ("Red", "Red") will not be flagged as an error.

However, logical errors across atomic expressions, such as the following, *will not* be checked for:

- Is not from (>10, > 25): a constraint expression subsumes another
- Is from (<10, >10): incompatible constraint expressions, no value complies with the constraint

You cannot constrain individual elements of an array or list differently, but you can constrain all elements of an array or list to a set of values. In other words, a single constraint on the array or list applies to each member of the array or list. Constraints on arrays and lists are specified in a similar manner as for individual basic data types.

Attribute Declarations Using Constrained Data Types

Once a constrained data type has been defined, it may be used to declare the Type of an attribute. User defined constrained data types created in an Aion BRE application are automatically made available in the Type drop-down box of the Attribute editor.

Constraints Restrictions

The following conventions apply to using constrained data types in an Aion BRE application:

- An array can be assigned a value of NULL even if NULL is not included in the list of its constraints. For lists and the other four basic data types, this is not the case; if you want NULL to be a valid value you must specify NULL as one of the valid values in the constraint list.
- You cannot declare local variables of any of the four constrained data types or of types derived from these four.
- You cannot declare class/instance attributes to be of the four basic constrained data types; the type must be a type derived from any of these four constrained types.
- Method return types or arguments cannot be any of the four constrained types or types derived from these.

Operation of Constraints

This section describes the edit-time and runtime operation of constraints and runtime behavior when a constraint violation is detected.

Edit-Time Considerations

A constraint on an attribute is in effect at edit time. Constraint checking is applied to direct assignments and assignments made through the initial value field.

- Constraining the *Initial Value* Field

The default "Initial value" for any attribute is NULL. For constrained attributes, an edit time error message will be displayed if NULL is specified as an "initial value" but is not included in the list of constraints for that attribute. The only exception to this rule occurs in the case of arrays, where NULL will be allowed as an "initial value" even if it is not specified in the constraints.

- Effect on assignments and comparisons

Any assignment to a constrained attribute must use a value consistent with the specified constraint on that attribute, otherwise an error message will be displayed.

However, edit-time checks on comparison are not performed. For example, if `intAttrb` is an integer attribute constrained to be `<= 100`, the following comparisons will not be detected as errors at edit-time:

```
if intAttrb > 120. . . // Comparison will always be false at runtime
loop . . . breakif (intAttrb > 120) // Will allow an infinite loop!
```

Nor will constraints be enforced on For-loops controls. The following code will not signal an error at edit-time:

```
for intAttrb = 120 to . . . // Note: local variables are typically used
for // loop control
```

Runtime Considerations

Constraint violations are detected at runtime in the following situations:

- expression evaluation, for example (where `intAttrb` is constraint to be `<= 100`):

```
intAttrb = 5 * XYZ // where XYZ is any non-constant expression and its
// value, multiplied by 5 exceeds 100.
```

- return value from a method, for example:

```
intAttrb = XYZ_method(5) // where XYZ_method returns a values that exceeds
// 100 based upon its input argument.
```

- the attribute being passed as an output argument of a method, for example,

```
XYZ_method(out Quantity) // where XYZ_method sets the value of the output
// argument to be > 100
```

However, CA Aion BRE does not enforce constraints in for-loop assignments. For example, the following code will not be detected to be in error even at runtime and may lead to unpredictable results if the control attribute is used in the body of the loop (again, where `intAttrb` is constraint to be `<= 100`):

```
for intAttrb = 120 to . . .
```

Runtime Behavior

How CA Aion BRE indicates a constraint violation at runtime differs between the interpretive (debugging) environment and the production (compiled) environment. Also, constraint violation behavior is different when using certain methods of `attribepointer`.

Behavior at interpretive runtime is as follows:

- On Windows:

When the application is run interpretively, a constraint violation will be reported using a dialog box. The user may choose to continue with the execution of application. Execution will continue with the new value. This kind of continuation will allow for further debugging of code.

- On UNIX and Mainframe:

An error message will be displayed on the standard output and execution will be aborted.

On the other hand, at compiled runtime constraint violations will be written to the `aion.log` file under the Windows environment if tracing is turned off. Under Unix and on the mainframe, violations will be reported through standard output. In addition, an error message indicating the constraint violation will be written to the trace file if tracing has been turned on. In all cases, execution of the application will be aborted.

Finally, the following methods of `AttributePointer` return `FALSE` if the given attribute value violates a constraint on the attribute:

- `AttributePointer::AssignAttributeValue()`
- `AttributePointer::SetAttributeValue()`
- `AttributePointer::SetNullValue()`

In case of a constraint violation, there is no application abort or no error message displayed. It is the programmer's responsibility to handle the return value and process it accordingly.

Note: A value of `FALSE` could also be returned if the assignment fails for any other reason, such as a mismatch in the base type of the attribute and the value.

Copy and Paste Objects

You can copy and paste objects within the same application and between different applications.

For step-by-step procedures for copying and pasting an object, see [Copying and Pasting Objects](#) in the CA Aion BRE online help.

For step-by-step procedures for copying, and pasting controls, see [Copying and Pasting Controls](#) in the CA Aion BRE online help.

Delete Objects

You can delete application objects from the following locations:

- Class Editor
- Explorer
- Output Window
- Project Workspace

For step-by-step procedures for deleting an object, see [Deleting Objects](#) in the CA Aion BRE online help.

For step-by-step procedures for deleting controls, see [Deleting Controls](#) in the online *Procedures*.

Edit Toolbars

CA Aion BRE provides a number of ready-made toolbars to give you quick access to common editing tasks. You can change the set of toolbars displayed during a development session and the set of tools displayed on a toolbar. You can also create your own toolbars.

Like the Output Window and the Project Workspace, toolbars can be docked (horizontally or vertically) or floated, and they can be dragged outside the Aion main frame window. By default, toolbars are docked to the top of the main frame window.

To display balloon help that describes a tool's function, place your cursor over the tool, and leave it there for a moment.

See [Using Edit Toolbars](#) in the CA Aion BRE online help for step-by-step procedures for performing the following operations:

- Relocating a toolbar
- Adding a tool to a toolbar
- Removing a tool from one toolbar and adding it to another
- Deleting a tool from a toolbar
- Restoring a toolbar's original set of tools

For step-by-step procedures for creating a new toolbar, see [Creating a New Toolbar](#) in the CA Aion BRE online help.

More Information:

[View Applications](#) (see page 91)

[Customize Toolbars](#) (see page 122)

Supplied Toolbars

The toolbars described in this section are standard.

Data Create Toolbar

Use the Data Create tools to add database connections, queries, and stored procedures to your application.



Decision Table Toolbar

Use the Decision Table tools to order rows in, and to add conditions and actions to a decision table.



Edit Toolbar

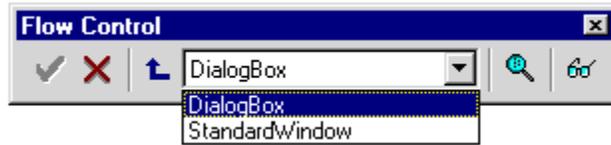
Use the Edit tools to undo, cut, copy, and paste the following items:

- Selected objects in the Project Workspace, the Explorer, and the editors
- Selected text strings in edit control fields (such as name, comment, or method body fields)



Flow Control Toolbar

Use the Flow Control tools to open, save, and close editors and to explore, find, and open objects.



When a class is open in an editor, the Flow Control toolbar gives you quick access to the class's base class and subclasses. To open its base class, click the blue L-shaped arrow icon. To open its subclasses, select them from the drop-down list box. The subclasses in the preceding figure are displayed when WinLib's `FrameWindow` class is open.

Layout Toolbar

Use the Layout tools to carry out various aligning, sizing, spacing, and ordering tasks in the Window Editor.



Menu Create Toolbar

Use the Menu Create tools to add menu items to menu titles and tool items to toolbars.



Method Editor Toolbar

Use the Method Editor tools when formatting and parsing logic statements in method bodies. These tools enable you to tab lines left and right, comment lines out, paste in operators and constructs, and find and replace text within a method body.



Object Create Toolbar

Use the Object Create tools to add classes, attributes, methods, and instances to your application.



Other Create Toolbar

Use the Other Create tools to add data types, associations, and interfaces to your application.



Query Editor Toolbar

Use the Query Editor tools to add field attributes, markers, and calculated fields and to invoke the SQL paster and the query test facility.



Standard Toolbar

Use the Standard tools to create, open, close, save, run, debug, and build applications; to show and hide the Project Workspace and Output Window; and to open the CA Aion BRE online help.



Window Create Toolbar

Use the Window Create tools to add standard windows, dialog boxes, control groups, menu titles, toolbars (runtime), and ActiveX controls to your application.



Include and Remove Libraries

You must use the Included Library Editor for the following tasks:

- To include a supplied or custom-built library in an application
- To remove an included library from an application
- To view information about an application's included libraries

To open the Included Library Editor, choose Edit Library Includes from the File menu.

Note: The Edit Library Includes option is available only for applications that have no unsaved changes. If this option is disabled, choose Save from the File menu to enable it.

Included Library Editor Fields

Use the fields in the Included Library Editor as follows:

- The Included Libraries field lists the libraries that have been *directly* included in the application. To view information about a library or to remove a library from the application, you must first select it in the Included Libraries field.

Libraries that appear beneath the Libraries node on the Project Workspace's Libraries tab page but that do not appear in the Included Libraries field were not directly included in the application. Instead, they piggybacked in as an included library of one of the libraries listed in this field. When you select a library in the Included Libraries field, a list of its included libraries (that is, the piggybacked libraries) is displayed beside the Included Libraries label in the Library Specific Information field (see the next bulleted item).

- The Library Specific Information field lists the selected library's included libraries and displays all text entered on the Comments tab page of the selected library's Library Properties dialog.

The Run Compiled option specifies whether to run the included library's built version (checked) or its interpreted version (unchecked) when you run the application. If you run the built version, you cannot debug into the included library at runtime.

- The Included Library Path fields specify the directories in which an application's included libraries (whether custom-built or supplied by Aion) are located. These fields can include multiple directories. Place a semicolon (;) between paths as follows:

```
C:\Program Files\CA\AionBRE\init;c:\libs
```

System-The paths entered in this field are used by all applications opened on your computer, and they persist from one session of Aion to the next.

Note: The information contained in the System field can also be modified in the Included Library Path field of the Settings dialog.

Application-The paths entered in this field are written to an application's .app file and are used only by that application. This feature enables you to transfer an application to other users without their needing to know the location of your application's included libraries (provided they are using the same network, drive names, and directory structure as you).

For step-by-step procedures for including a library in or removing a library from an application, see Including and Removing Libraries in the CA Aion BRE online help.

More Information:

[Customize the Development Environment](#) (see page 89)

Open Applications That Have Included Libraries

When you open an application that has included libraries, Aion searches for the included libraries in the following three locations, checking these locations in the order listed:

- The directory in which the application is located (that is, the “current” directory)
- The Application field of the Included Library Editor
- The Included Library Path field of the Settings dialog

Keep this search order in mind when deciding where to locate an application's included libraries. If several libraries on your system have the same name, Aion uses the first of those libraries that it finds when searching for an application's included libraries.

Note: If an application's included libraries are not stored in one of these three locations, the application either will not open or will open with invalids.

More Information:

[Customize the Development Environment](#) (see page 89)

Libraries Included with Aion BRE

CA Aion BRE ships with the following libraries installed.

- actxlib.app
- autolib.app
- cobslib.app
- complib.app
- datalib.app
- domlib.app
- dynrdlib.app
- dynrelib.app
- dynrlib.app
- empty.app
- guiv7lib.app
- iolib.app
- iowlib.app
- mqlib.app
- reclib.app
- rmlib.app
- saxlib.app
- starter.app
- syslib.app
- varlib.app
- winlib.app
- xmllib.app
- xsdlib.app

Work with Source Control

Source control programs safeguard application code in a multideveloper environment by permitting only one developer at a time to modify application files housed in a source control program. Source control programs also save and store sequential versions of application files, preserving your code at various stages of development and enabling developers to review the history of a file's changes and to revert to an earlier version of the file if necessary.

Aion BRE works with any third-party source control program that conforms to the SCC API standard established by Microsoft.

Note: For information about installing a particular source control program, see the documentation provided by the program's supplier.

You can add an entire Aion BRE application to a source control program by adding only the application's .app file or files and any graphic resource files that it uses (such as .bmp, .cur, .ico, and .stg directory files).

Note: To facilitate simultaneous development under source control, you should divide your application into smaller libraries. Aion will generate an .app file for each library. This allows developers to check out an .app file that encompasses just one part-or module-of the application without locking other developers out of their respective modules.

More Information:

[.APP and .BIN Files](#) (see page 80)

Set Source Control Safeguards

CA Aion BRE provides optional prompts to help keep your interaction with a source control program error-free. You set these prompts in the Source Control Options dialog.

To open the Source Control Options dialog, from the Tools menu, select Options, Source Control.

The Source Control dialog options are global: they affect all applications running on your computer; their values are stored in your system's registry; and they persist from one session to the next.

- To display a dialog that lets you check out a file from source control if you try to edit the file while it is checked in, select the *Check out source file when edited* option.
- To display a dialog that lets you check files in to source control if they are checked out when you close Aion, select the *Check in file when closing the application* option.
- To display a dialog that lets you save an application if you try to check the application in to source control without saving its changes, select the *Prompt to save application before source control operation* option.

Note: By default, Aion prompts you to save the application before and after source control operations. You should not disable this prompt.

- To display a dialog that lets you restore an application if you check the application's .app file out to a directory containing the application's out-of-date .bin files, select the *Prompt to restore application after source control operation* option.
- To see additional options (if any) supported by your source control program, click *Advanced*. (If the source control manager that you use does not support additional options, this button is disabled.)

When Advanced is clicked, the source control program displays its Preferences dialog and is responsible for getting and for maintaining the persistence of information in that dialog.

Source Control Menu Options

The Source Control option on the Tools menu supports the following source control program operations. Depending on the state of an application, some options may be unavailable at certain times. In addition, not all source control programs support all these options.

Operation	Result
Get Latest Version	Copies the latest version of the .app file from source control into the current working directory but does not prevent the master copy of the .app file from being checked out and edited by other users.
Check Out	Copies the latest version of the .app file from source control into the current working directory and makes the master file read-only for all other users.
Check In	Copies the checked out version of the .app file from your working directory to source control, creating a new version of source control's master file.
Undo Check Out	Nullifies a check out that has not been checked in, canceling all changes made since the file was checked out.
Add to Source Control	Copies the selected application's .app file to source control if the .app file is not already in source control.
Remove from Source Control	Deletes the currently selected application's .app file from source control.
Show History	Displays a list of all the versions of the .app file stored in source control.
Show Differences	Displays the differences between a version of the .app file in source control and the corresponding .app file in your working directory.
Source Control Properties	Displays information about the .app file in source control.

Operation	Result
Share from Source Control	Copies a file from another source control project into the current project, creating a link between the two projects. Checking the file in to or out of either project checks it in to and out of both projects.
Refresh Status	Updates the file status display in the source control window (which tracks file changes made by all source control users).
Source Control Browser	Finds the source control program, and opens it in Aion.

For step-by-step procedures for adding an application to Source Control and for checking an application in and out of Source Control, see Working with Source Control in the CA Aion BRE online help.

Enable Concurrent Development

If you want multiple developers to work on the same .app file at the same time and then to merge their changes in source control, you must set the appropriate option in the source control program when you install it. You cannot enable this option through Aion.

For more information about enabling concurrent development through source control, refer to the documentation that came with your source control program.

Change Management

Source control programs coordinate development at the *component level* by allowing only one developer at a time to work on an .app file. Change Management, on the other hand, tracks, documents, and controls development at the *object-property level*, enabling multiple developers to work concurrently on the same .app file and then to merge their changes.

For step-by-step procedures for working concurrently on the same .app file, see Working with Change Management in the CA Aion BRE online help.

For step-by-step procedures for performing additional change management operations, see the following topics:

- Set the Baseline
- View Change Files
- Create a Change File (see CA Aion BRE online help)
- Save Changes to a File
- Apply Change File

Change Management Functions

The Change Management functions, located on the Tools menu, Work Group option, perform the following tasks:

- Track changes made to object properties in application files
- Enable developers to document why changes have been made
- Display records of changes for online viewing
- Save records of changes for archiving, printing, and applying to files
- Apply changes to application files, reporting any conflicts that arise

The following sections explain how to use the Change Management functions.

Set the Baseline

A *baseline file* is a copy of an .app file that preserves the .app file's code at a given moment of development, creating a gauge against which Aion can track subsequent changes made to the properties of the objects in the .app file. To begin tracking modifications with Change Management, you must generate a baseline file for the .app file.

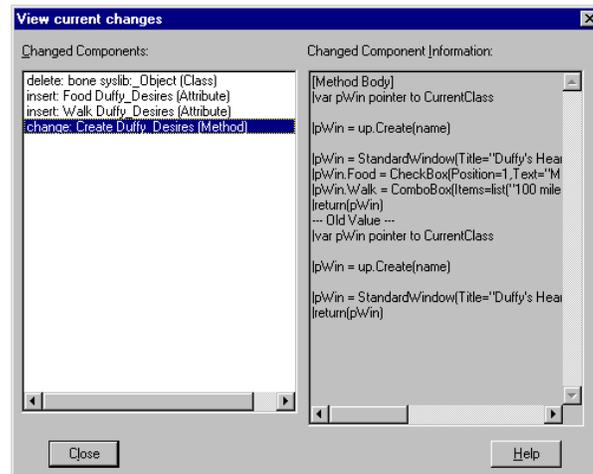
For step-by-step procedures for setting the baseline file, see Setting the Baseline in the CA Aion BRE online help.

View Changes

Once you create a baseline file, you can use the View Changes function of Change Management to see the differences between the baseline file and the current state of your .app file.

To view differences between the baseline and the current file

From the Tools menu, choose Work Group, View Changes. The View Current Changes dialog opens:



Note: To display changes made to an .app file, you *must* set a baseline for the .app file *before* you modify it. You do not, however, need to create a change file for it.

If you select View Changes before setting a baseline or, if you previously set a baseline, before making any changes to a file, Aion displays the message *No Changes in Change File* instead of the View Current Changes dialog.

For more information, see Viewing Change Files in the CA Aion BRE online help.

Changed Components Field

The Changed Components field of the View Current Changes dialog lists all the objects (that is, classes, methods, attributes, and instances) that have changed in the .app file since the baseline file was last generated. In addition, if you add or remove an included library, this field displays an item that compares your application's baseline set of included libraries with its current set.

Each item in the list is prefaced by one of the following identifying words:

- **Insert**-identifies an object that was added to the .app file after the baseline was set.
- **Delete**-identifies an object that was removed from the .app file after the baseline was set.
- **Change**-identifies an object that was modified after the baseline was set.

The information following the identifier consists of (1) the object's name, (2) its owning class, and (3) its type. For example, the following entry indicates that a method named Create and owned by the class AppWindow was modified after the baseline was set:

change: Create AppWindow (Method)

Changed Component Information Field

When you click an object in the Changed Components field, details about the properties of the object that were changed are displayed in the Changed Component Information field. The current state of the property displays first, and the baseline state displays beneath the heading "Old Value."

Save Changes to a File

To preserve a record of the changes made to an .app file, you must save them to a *change file*. Change files are ASCII text files that contain the results of a comparison between the current state of an .app file and its baseline. Change files list all objects and object properties that have been added, revised, or deleted in the .app file since the baseline (or .bsl) file was last generated.

For step-by-step procedures for creating and viewing change files and applying change files, see Working with Change Management in the CA Aion BRE online help.

Apply Change File

Use the Apply Change File function to incorporate a change file's changes into an .app file. If you have change files from several independently modified copies of an .app file, you can use this function to merge those change files into a single .app file.

For step-by-step procedures for creating, saving, and viewing change files and applying change files, see Applying Change Files in the CA Aion BRE online help.

Manual Repairs

When applying multiple change files to a master .app file, be aware that a change made to an object property in one change file can overwrite a change made previously to the same property by another change file. To avoid doing a lot of manual repair work to fix overwritten changes, it is a good idea to review the changes in *all* change files before applying any of the change files to the master. You can then order the application of the change files to avoid canceling out as many changes as possible.

For example, if change file A adds a significant amount of code to a method while change file B contains a one-line change to the same method (among other changes) and you want to incorporate both method changes into the master file, applying file A after file B will minimize the repair you must make to the master file to get both changes into the final version of the method.

Produce Reports for an Application

CA Aion BRE provides facilities for producing reports about the contents of a knowledge base. These include reports on:

- Specific objects
- An entire library
- A single class
- Aion documentation, including the business logic of the application (the applications static rules and decision tables)

The Application Print utility provides the ability to print hardcopy reports about specific objects, an entire library, or a single class. You can define the content of these reports in the Application Print Utility dialog.

For step-by-step procedures for configuring the application print utility to print hard-copy reports directly from the knowledge base, see the following topics in the CA Aion BRE online help:

- Printing Reports for a Specific Object
- Printing Reports for an Entire Library
- Printing Reports for a Single Class

Generating Aion Documentation requires special considerations.

More Information:

[Printed Report Contents](#) (see page 134)

[CA Aion BRE Documentation](#) (see page 134)

Printed Report Contents

Reports printed using the Application Print utility are structured as follows:

- The header-specified in the Application Print Utility dialog-appears at the top of each page.
- The first item in a global report is the application's name, which appears in boldface type. This item does not appear in local reports. If you check the Detail Library Info option, information about an application's owner, build, and revisions follows its name.
- In global reports, a list of included libraries follows the application information. This list of libraries does not appear in local reports. If you check the Detail Library Info option, information about a library's owner, build, and revision follows each library name.
- In global reports, a list of all the application's classes appears after the list of included libraries. This list does not appear in local reports.
- Finally, a section devoted to each listed class appears. This is the first and only section of local reports which contains information about just one class. Comments and information about the class's members (attributes, constants, static instances, and methods) and about its base class appear in these sections. If you do not check the Header Format option, detailed information about each of the class's methods is also printed.

For step-by-step procedures for printing an application report, see Printing an Application Report in the CA Aion BRE online help.

CA Aion BRE Documentation

The CA Aion BRE Documentation (AionDoc) report provides customizable documentation of an Aion BRE knowledge base in an HTML format. The advantages of the Aion Documentation report are that it can include multiple libraries and it can be focused to produce documentation for a specific purpose. For example, the report can be customized to report just the exported classes of an application (the interface of the Aion BRE component) or to report the business logic of the application from the perspective of management and domain experts.

The CA Aion BRE Documentation report is generated by selecting the Generate AionDoc option on the File menu. Reporting options provided by the AionDoc wizard allow you to generate documentation for

- Any or all libraries contained in the application
- The exported classes in the application
- The datatype, class, and interface hierarchies
- The business logic of the application (the rules and decision tables of the application)
- The implementation of methods
- Any static objects
- The rule network

To remain in the dialog and enable the Close Wizard button for optional use, uncheck the *Close Wizard at Close* checkbox.

For a step-by-step procedure for generating an CA Aion BRE Documentation Report, see *Generating CA Aion BRE Documentation* in the CA Aion BRE online help.

The CA Aion BRE Documentation report includes indexes for Libraries, Classes, and Rules in an application. The Libraries report contains the complete class hierarchies, up through SysLib, of the classes in the selected library; the Classes report contains documentation on each class in the libraries selected in the AionDoc wizard according to the desired options; the Rules report contains the documentation and content of the rules methods in the selected libraries according to the desired reporting options. Rich hypertext links in each report allows convenient navigation.

The CA Aion BRE Documentation HTML-file must be linked with a cascading style sheet. To view the report with a predefined style sheet, *aiondoc.css*, the AionDoc output path should point to the directory containing *aiondoc.css*. By default this directory is the \AionDoc subdirectory within the Aion install directory. The *aiondoc.css* file can be customized by users familiar with cascading style sheets. The HTML file that Aion generates can be viewed and printed with Internet Explorer 5.

Note: Aion supports the use of HTML tags in Class and Rule comments. These tags will be used when formatting these comments in the Aion Documentation report.

Search for Objects by Name

The Look Up Object utility helps you find a single object quickly. This utility is useful when you know the name of the object that you want to find but cannot remember what class or included library it belongs to.

For step-by-step procedures for using the Look Up Object utility, see Searching for Objects by Name in the CA Aion BRE online help.

Search for Objects by Multiple Criteria

Use the Find utility to search for objects or text strings in an application and its included libraries. This utility is especially useful for finding groups of related objects. For example:

- To list all classes containing "win" in their names, enter *win* in the Name field, and check Class in the Types field.
- To list all methods whose names begin with "When," enter When* in the Name field, and check Method in the Types field.
- To list all objects containing the text "create," enter create in the Search String field, enter an asterisk (*) in the Name field, and check Any in the Types field.

You can also use the Find utility to get a count of objects by type.

Example:

- To see how many attributes the open application contains, leave the Search String field blank, enter an asterisk (*) in the Name field, check Attribute in the Types field, and select Global. On the Results tab page of the Output Window, the list of attributes returned from the search will be grouped under a header stating how many attributes were found. For example,
Attribute in all classes defined in local library {97 objects found}

For step-by-step procedures for using the Find utility, see Using the Find Utility in the CA Aion BRE online help.

Replace Text

Use the Replace utility to find and replace text in an application.

For step-by-step procedures for using the Replace utility, see Replacing Text in the CA Aion BRE online help.

Search Across Applications

The Find Library utility enables you to search for objects that reside in applications other than the one in which you are working. This utility searches through all applications located on a specified path, looking for objects by name, looking for keywords in Comment properties, or looking for objects that contain the specified search string in *any* of their properties.

For step-by-step procedures for using the Find Library utility to search across applications, see Searching Across Applications in the CA Aion BRE online help.

More Information:

[Create a Graphical User Interface](#) (see page 139)

Chapter 5: Create a Graphical User Interface

CA Aion BRE provides tools for building a graphical user interface (GUI). This chapter explains how to use those tools to create windows, controls, and graphics for your application. This chapter also discusses techniques for specializing the GUI objects supplied by CA Aion BRE to fit your application's needs, and how to integrate your GUI objects with application logic.

For additional information on specific GUI objects and the classes from which they are derived, see the "WinLib" chapter in the CA Aion BRE online help.

This section contains the following topics:

[How You Create a GUI](#) (see page 139)

[Work with Object Properties](#) (see page 141)

[Work with Windows and Dialog Boxes](#) (see page 146)

[Add Controls to Windows](#) (see page 151)

[Add Menus to Windows](#) (see page 163)

[Add Toolbars to Windows](#) (see page 171)

[Add Graphics to Windows](#) (see page 175)

[Implement Logic to Run Your GUI](#) (see page 178)

[Conclusion](#) (see page 186)

How You Create a GUI

In CA Aion BRE, there are four main steps to creating a graphical user interface. After completing Step 1 and mapping out a design for your GUI, refer to the following sections of this chapter to learn how to complete Steps 2, 3, and 4.

Step 1-Define data requirements.

Whether your application gathers its data from external databases, from user input, or from a combination of these sources, you should define the graphical requirements for working with that data before you begin creating your interface.

Step 2-Construct the application's main frame window.

New Aion BRE applications come with a main frame window named AppWindow and the code required to create and open this window at runtime. AppWindow contains no menus, toolbars, controls, or graphics, so you must create these objects and attach them to AppWindow using the following tools:

- Menu Editor to design menu titles and menu items
- Tool Editor to design toolbars
- Window Editor toolbar to add controls and graphics
- Screen menu to design, attach, and align controls
- Color and font tools to refine the appearance of the main frame window and its elements
- Properties dialog boxes to change text and other features of the main frame window and its elements

More Information:

[Stand-Alone Aion BRE Applications](#) (see page 45)

Step 3-Create dialog boxes.

Each dialog box that you create is displayed in the Window Editor. When a dialog box is open in the editor, use the tools listed in Step 2 to customize it.

Step 4-Add logic.

Logic provides the network of decisions that run your application. GUI objects require logic to perform the following tasks:

- Opening AppWindow when the application starts.
- Loading values into controls on the main frame window.
- Opening dialog boxes. Such logic is normally included in an event-triggered method attached to a menu item, push button, or toolbar.
- Enabling controls on a dialog box to interact with other controls (choosing one control enables another, and so forth).
- Validating and processing user input in a dialog box. Input can be accepted by controls on the dialog box.
- Manipulating and saving collected data.

Work with Object Properties

In CA Aion BRE, each GUI object has multiple characteristics, or *properties*, that describe the object. For example, a push button has properties such as Attribute Name, Owning Class, Style, and Text (among others). A push button designed to process user input on a dialog box might have the following values in those properties:

- Attribute Name: EnterInput
- Owning Class: PushButton
- Text: Enter
- Style: Default Push Button

You set the values for an object's properties in the object's Properties dialog box. All GUI objects—including windows, dialog boxes, menu titles, menu items, toolbars, tool items, controls, and graphics—have Properties dialog boxes.

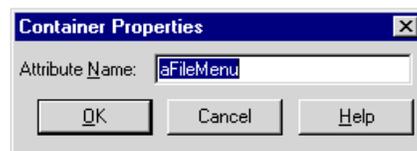
Open and Use Properties Dialog Boxes

This section discusses properties that are common to most interface objects. To learn about properties that are specific to specific objects and to learn how to open an object's Properties dialog box, see one of the following sections:

- Standard Window and Dialog Box Properties
- Control Properties
- Menu Properties
- Toolbar Properties
- Graphic Properties

GUI Properties versus Container Properties

If you double-click a menu title, toolbar, or control group in the Window Editor of its parent window, the clicked control's Container Properties dialog box, not its Properties dialog box, opens:



The Container Properties dialog box enables you to change the attribute name of the control. A control's attribute name appears beneath its owning class in the Project Workspace and on the Members tab page of its owning class's Class Editor. The Properties dialog box, on the other hand, enables you to set the control's runtime characteristics, which are the topic of this discussion.

Common Object Properties

Some properties are common to many or all application objects. For example, all objects have an Attribute Name property. The following sections describe common object properties.

Attribute Name

This is the object's system name. If you do not enter a name in this field when you first save the object, a name is automatically assigned to the object. Use this name to refer to the object in application logic (for example, `AttributeName.SetEnabled(true)`). When you create event-triggered logic for the object, the system uses this name in method titles such as *WhenAttributeNameChosen* and *WhenAttributeNameSelected*.

Note: If you change the Attribute Name, references to the old name in the application logic become invalid. To find and validate invalid references, use the Invalids tab page in the Output Window.

Basic Window Styles

The window styles that you assign to a control determine aspects of both the control's appearance and its behavior. There are four basic window styles:

- **Disabled**-Use to disable a control when its parent window first opens. Typically, a disabled control is enabled when a user chooses an appropriate option with another control. To enable a disabled control at runtime, use the `SetEnabled` method.
- **Group**-Use to gather controls into groups that function (in some ways) as one control. When you assign the Group style to a control, all the controls that follow it-as listed in the Order Controls dialog box-become part of that control's group. The group ends at the next control assigned the Group style.
 - For example, if controls A, B, C, and D are listed alphabetically in the Order Controls dialog box and you assign the Group style to controls A and D, controls A, B, and C make up one group.
 - When controls are grouped, users can move keyboard focus among the controls in the group with the arrow keys. When buttons are grouped, users can choose only one radio button per group.
 - From the Window Editor Screen menu, choose Order to display the Order Controls dialog box.
- **Tab Stop**-Use to halt keyboard focus at a control when a user moves focus among a window's controls with the tab key. When a user hits the tab key, focus jumps to the next control-as listed in the Order Controls dialog box-that has the Tab Stop style set. Controls without this style are passed over.
- **Visible**-Use to display a control when its parent window first opens. This is the default for all controls.
 - To make an invisible control visible (or vice versa) at runtime, use the `SetVisible` method.

Color

Many Properties dialog boxes include a Color push button. Click this to open the Color dialog box, where you can set foreground (text) and background colors for a control.

Owning Class

This is the name of the class from which an object is instantiated. The owning class can be the original object class supplied in WinLib or a developer-defined subclass.

Pointer

This is the cursor shape displayed when a user moves the mouse pointer over a control. When you click the Pointer push button, the Open Cursor dialog box displays a list of predefined mouse pointer resources in the Defined Pointers field. Select one of these, or use the list boxes for Directories and Pointer Files to create a cursor resource from another .cur file. If you do not select a pointer for this field, the predefined cur_normal arrow cursor is used by default.

Note: The Pointer field must contain the name of a resource's static instance, not the name of the *.cur file from which the resource was created. For more information, see Create Resources.

Text/Title

Text specified in a Text or Title field is displayed on a control. This property is available with controls:

- Check boxes
- Group boxes
- Push buttons
- Radio buttons
- Static text
- Tab pages
- Text windows

Note: To display text with other controls, create a label using a static text control.

Choose Fonts

You can use any font available through Windows on your interface objects. If your application later runs on a computer that does not have the same font, Windows substitutes the closest available font. To be reasonably sure that your choice is always available, choose from the fonts provided by Windows.

For step-by-step procedures for choosing a font in the Window Editor, see Choosing Fonts in the CA Aion BRE online help.

Choose Colors

Normally, the user's system determines interface object colors. Windows users select a color scheme provided with Windows or create their own custom colors, and these colors are used in all applications. In Aion BRE applications, however, you can override the user's default system colors for some GUI objects.

Resetting a color is not recommended for general use. It can, however, help emphasize a critical control or message. Colors can also be useful when working with graphics. The Color dialog box offers you a wide range of colors and lets you define custom colors.

The effect of changing the text color depends on the type of control you choose. For controls that do not display text at edit time—such as combo boxes, list boxes, and many text windows—the text colors you choose apply to text entered or displayed at runtime.

For step-by-step procedures for choosing colors in the Window Editor or in the Color dialog box, see [Choosing Colors](#) in the CA Aion BRE online help.

Scope of Color Change

The extent to which you can change a control's color varies from control to control as described in the following table:

Control Type	Color Change
Check box Radio button Static text	Text, outline, and background change.
Combo box Edit window Group box List box Text window	Text and background change. Outline remains black.
Image list Tree list	Only background changes.
Image button Progress bar Push button Scroll bar Slider Tab Up-down	Nothing changes—color option is unavailable.

Work with Windows and Dialog Boxes

Often, an application contains one standard window and many dialog boxes. The standard-or *main frame*-window frames the application and serves as a gateway into it. The dialog boxes, which are also a type of window, open from this main frame window, prompting users for input or displaying messages. Multiple Document Interface (MDI) applications, which require a standard window for each displayed view, are an extension of this single standard window design.

More Information:

[Multiple Document Interface](#) (see page 148)

Subclass Supplied Classes

CA Aion BRE provides two classes, `StandardWindow` and `DialogBox`, from which you can create windows and dialog boxes for your GUI. These classes are derived from the `FrameWindow` class in the WinLib supplied library.

`StandardWindow` and `DialogBox` are bare-bone containers designed to serve only as base classes. When you create a window or dialog box using the Aion BRE visual GUI-building tools, a subclass derived from one of these supplied classes is automatically generated for you. This subclass inherits all the members (attributes, constants, and methods) of its base class.

Using the tools provided in the Window Editor, you can customize a subclass by modifying its inherited members to suit your application's needs. After customizing a subclass, you can then use it as a base class for creating other windows or dialog boxes.

Note: Developer-created subclasses are added to the application itself, not to an included custom or supplied library, such as WinLib.

Example:

The `AutoDialogBox` class in the WinLib supplied library is an example of a subclass that is used as a base class. When you create a dialog box in Aion BRE, you can choose to derive it from either the `DialogBox` or the `AutoDialogBox` class. `AutoDialogBox` itself, however, is derived from `DialogBox`. As a subclass of `DialogBox`, `AutoDialogBox` shares most of `DialogBox`'s characteristics, but it also has three extra methods (`Create`, `WhenCancelChosen`, and `WhenOKChosen`) and three push buttons (`Cancel`, `Help`, and `OK`) that are not part of `DialogBox`. Thus, any dialog box derived from `AutoDialogBox` inherits not only the standard characteristics of `DialogBox` but also the distinguishing characteristics of `AutoDialogBox`.

More Information:

[Overview of CA Aion BRE Objects](#) (see page 43)

StandardWindow Class

Windows derived from the StandardWindow class can use drop-down menus to display available options. When a user selects an option from a standard window menu, the logic initiated may act on information displayed in that window or it may open a dialog box and perform its tasks there. No matter where the logic is carried out, however, control returns to the standard window when the logic concludes.

Note: Although the Window Editor allows you to add menus to child windows (standard windows with the Child style set), child windows cannot have menus. At runtime, a child window that contains menus does not display a menu bar.

The default standard window is nonmodal. Users can shift focus from a *nonmodal* window to another window without first exiting the nonmodal window. This allows users to work in other windows while the nonmodal window remains open.

AppWindow, the default main frame window for all new Aion BRE applications, is derived from StandardWindow.

DialogBox Class

Use the DialogBox class to create dialog box windows that open in response to user actions (such as selecting a menu option or clicking a button) and that display, or prompt users to input, a related set of information.

The default dialog box is *modal*. Because modal dialog boxes must be closed before the application can continue, they force the user to complete and process the tasks that they contain.

AutoDialogBox Class

AutoDialogBox is a subclass of the DialogBox class. All dialog boxes derived from AutoDialog contain three standard push buttons:

- OK
- Cancel
- Help

And the following three predefined methods:

- Create
- WhenCancelChosen
- WhenOKChosen

As defined, the WhenOKChosen method automatically validates user input when the Auto Validation property of a control in an autodialog box is enabled.

AutoDialogBox has been added to the WinLib for your convenience. If you prefer, you can construct your own input-validating subclass of DialogBox, adding the necessary buttons and writing your own versions of the WhenChosen method.

More Information:

[Auto Validation](#) (see page 162)

Multiple Document Interface

You can create either Single Document Interface (SDI) or Multiple Document Interface (MDI) windows with Aion BRE. Deciding which to use is a design decision that depends on the data your application must handle. The default style for a standard window created with Aion BRE is SDI.

MDI functionality enables you to display multiple standard windows in an application simultaneously. Each of these windows is known as an MDI child window. The child windows are contained in an MDI parent window. Users can open several MDI child windows at the same time without reloading the parent window. They can switch between the open child windows and work in whichever window is active. Child windows can be moved to the edge of a parent window, but they cannot be displayed outside it. (See, for example, the Aion BRE editors, which are MDI child windows of the Aion BRE main frame window.)

A typical MDI application is composed of a main frame window marked as the parent window and a number of child windows that open from the parent.

In CA Aion BRE, MDI windows come with methods that perform the functions found in the Window menu of most Windows applications. To use these methods, add a Window menu containing various MDI display options (such as Cascade, Tile Horizontally, and so forth) to your MDI application's menu bar, and then attach the MDI methods that Aion provides to these display options.

To learn how to create MDI windows, see To create an MDI parent or child window in the Window Type section in the CA Aion BRE online help.

To learn how to create menu titles and menu items, see Creating Menu Titles in the CA Aion BRE online help.

For more information about MDI methods, see the StandardWindow Methods section in the "WinLib" chapter in the CA Aion BRE online help.

Create Windows and Dialog Boxes

The New Standard Window dialog provides necessary fields for defining a new window for a GUI.

The New Dialog Box dialog provides necessary fields for defining a new dialog for a GUI.

For step-by-step procedures to create windows and dialog boxes, see Creating Windows and Dialog Boxes in the CA Aion BRE online help.

Edit Windows and Dialog Boxes

To edit a window or dialog box in the Window Editor, use the procedures that follow.

Note: In CA Aion BRE both the StandardWindow class and the DialogBox class are derived from the FrameWindow class. Thus, editing procedures are usually the same for both windows and dialog boxes. In the following procedures, the word *window* stands for both window and dialog box.

See the following topics in the CA Aion BRE online help for step-by-step procedures for performing these operations:

- Resizing a Window
- Repositioning a Window
- Previewing a Window
- Deleting a Window

Standard Window and Dialog Box Properties

When you create a new standard window or dialog box, it comes with a set of default property values. To review or modify these values, open the window or dialog box's Properties dialog.

For step-by-step procedures to open a window or a dialog box's Properties dialog box, see Standard Window and Dialog Box Properties in the CA Aion BRE online help.

Common Properties

The StandardWindow and DialogBox Properties dialog boxes contain three tab pages: General, Style, and Title.

More Information:

[Common Object Properties](#) (see page 142)

Style Properties

By default, standard windows can be resized at runtime but dialog boxes cannot. To change these sizing defaults, select the appropriate Style option:

- **Dialog Border**-makes the window or dialog box unsizeable at runtime.
- **Size Border**-makes the window or dialog box sizeable at runtime.

Initial State

The value for the Initial State property determines how a standard window looks when it is first opened. This property does not apply to dialog boxes.

You can choose among three styles when setting a standard window's Initial state:

- **Maximized**-covers the entire screen area when it opens. (A maximized MDI child window covers the entire work area of the main frame window.)
- **Minimized**-first displayed as an icon.
- **Normal**-opens to the size set in the Window Editor. You set this size in the editor by dragging the window's sizing handles.

Window Type

Use this property, which applies only to standard windows, to create either Single Document Interface (SDI) or Multiple Document Interface (MDI) windows.

For step-by-step procedures for creating an MDI parent or child window, see Window Type in the CA Aion BRE online help.

Add Controls to Windows

Controls are the interface objects that sit on top of a window, displaying values, prompting for input, or both. Technically speaking, controls are windows. The relationship between a container window and the control windows that it contains is often referred to as a *parent-child* relationship, the container being the parent and the controls being the children. In Aion BRE, a control is an attribute of its parent window's class.

CA Aion BRE provides a wide range of common Windows controls to help you design GUIs that have the look and feel of Windows. All Aion BRE controls are derived from the WindowObject class, which is located in the WinLib supplied library.

There are two types of controls in Aion BRE:

- Container controls

Container controls are controls to which other controls can be attached. There are only six container controls in Aion BRE: control groups, menu titles, splitter windows, tab controls, tab pages, and toolbars. For more information about these controls, see the CA Aion BRE online help following sections:

- Working with Control Groups
- Creating Menu Titles
- Working with Splitter Windows
- Working with Tabs
- Creating Toolbars

- Controls that cannot contain other controls

All controls supplied by Aion BRE except those mentioned above fall into this category.

Controls Supplied by CA Aion BRE

The following table provides a brief description of the Aion BRE controls. For more information about a particular control, see the “WinLib” chapter in the CA Aion BRE online help.

Control	Description
Check Box	A rectangular box used to turn on or off an option. When an option is selected, an X appears in the box. Check boxes can be displayed with labels.
Combo Box	<p>A list box combined with a text window. When users select an item in the list box, it is displayed in the text window.</p> <p>There are three types of combo boxes:</p> <p>Simple-The list box is always displayed. If the user types text in the window, the list box highlights the first selection that matches the entry.</p> <p>Drop-down-The list box drops down when the user selects the drop-down arrow next to the control. If the user types text in the window, the list box highlights the first selection that matches the entry.</p> <p>Drop-down list-The list box drops down when the user selects the drop-down arrow next to the control. To enter text in the window, users <i>must</i> select directly from the list; they cannot type text in the window.</p>
Edit Window	A window in which users can enter and edit text. Although similar to a Text Window control, it is designed to handle a larger volume of text.
Group Box	A frame or box that can be labeled and that encloses a set of related options. It is a visual device only; controls enclosed in a group box are not children of the group box.
Image Button	A push button labeled with a bitmap image instead of with text.
Image List	A window that displays a collection of icons with labels (as in the right panel of the Windows Explorer) or columnar lists of text with or without icons. An image list can have four different views: icon, small icon, list, and report.
List Box	<p>A window that displays a list of items that users can select, such as a single-column list of filenames or a multicolumn, labeled list of class instances and their respective attributes.</p> <p>In a <i>single-selection</i> list box, users can select only one item.</p> <p>In a <i>multiple-selection</i> list box, users can select a range of items.</p>

Control	Description
	In a <i>checkbox</i> list box, users can select a variety of items, adjacent or not. Users cannot type a selection in a list box.
Menu Title	The root item and container window for menu bar drop-down menus.
Progress Bar	A rectangular window that is gradually filled with a color from left to right as an operation progresses, indicating the percentage of the operation, such as printing, that has been completed.
Push Button	A rectangular button, labeled with text, that initiates a command.
Radio Button	A round button used to select one of a group of mutually exclusive options (users can select only one button at a time in a group). When an option is selected, a black dot appears in the button.
Scroll Bar	A horizontal or vertical bar containing arrows that can be clicked and a box that can be clicked and dragged to scroll the contents of another control, such as a list box, text window, or combo box.
Slider	A window that contains an indicator on a gauge displaying optional tick marks. Users can move the indicator in specified increments to set a value—such as speed, brightness, or volume—from a continuous range. Also called a <i>trackbar</i> .
Splitter Window	A window divided either horizontally or vertically into two panes by a splitter bar. Users can resize the panes at runtime by moving the bar. Each pane can display one instance of any class descended from WinLib's Window class.
Static Text	A text field that can be used to label other controls. It takes no input from users.
Tab	A window whose display is analogous to dividers in a notebook or labeled folders in a file cabinet. It enables you to define multiple pages for the same area of a window or dialog box. Each page contains information or controls that a user can display by selecting the corresponding tab. Note: In Aion BRE, a tab control is composed of two types of container controls: a <i>tabbed window</i> (which can contain only tab pages) and a <i>tab page</i> (which can contain most other controls).

Control	Description
Text Window	A window that displays text and that can accept user input. Users can either select the displayed text or delete it and type new text. Compare Edit Window control.
Toolbar	A bar-like container window used to group tool items.
Tree List	A window that displays a hierarchical list of items. Each item consists of a label and an optional bitmapped image, and each item can have a list of sub-items associated with it. By clicking the + button associated with it, users can <i>expand</i> (display) and <i>collapse</i> (hide) the associated list of sub-items. The class list on the Libraries tab page of the Project Workspace and the left panel of the Windows Explorer are examples of tree list controls.
Up-Down	Also called a <i>spin button</i> . A pair of arrows that users can click to increase or decrease a value, such as a scroll position or a number displayed in a companion control (known as a <i>buddy window</i>). To the user, an up-down control and its buddy window often look like a single control. Up-down controls are used most often with text windows to allow the selection of numeric values.

Create Controls

When working in the Window Editor, use the Window Editor's toolbar and the menu bar's Screen menu to create and add controls to windows and dialog boxes.

See the following topics in the CA Aion BRE online help for step-by-step procedures:

- Creating Control Groups
- Creating Menu Titles
- Creating Splitter Windows
- Adding Tab Controls to a Window
- Adding Tab Pages to Tab Controls
- Adding Controls to Tab Pages
- Creating Toolbars
- Creating Other Controls

Edit Controls

Use the following tools to edit and arrange controls in the Window Editor:

- Window Editor toolbar
- Menu bar's Screen menu
- Layout toolbar

For step-by-step procedures for editing and arranging controls in the Window Editor, see the following topics in the CA Aion BRE online help:

- Displaying the Grid
- Selecting Multiple Controls
- Moving Controls
- Aligning and Spacing Controls
- Sizing Controls
- Copying and Pasting Controls
- Deleting Controls

Label Controls

Many controls, such as check boxes, group boxes, push buttons, radio buttons, and text windows, have Text or Title properties in which you can enter a label to describe the control's function. Providing descriptive labels is not mandatory, but doing so helps end users interact more efficiently with your application's interface.

For step-by-step procedures for labeling controls, see Labeling Controls in the CA Aion BRE online help.

Static Text

To create labels for controls that do not have Text or Title properties, use static text controls.

Add Mnemonics to Labels

Check boxes, image buttons, push buttons, radio buttons, and static text support mnemonics. To create mnemonics for these controls, enter an ampersand (&) in the control's Text field immediately in front of the character you want to assign the mnemonic to.

For example, entering &Sort in a button's Text field makes it possible to choose the radio button labeled Sort by pressing Alt+S when the dialog box that contains the radio button is in focus.

Order Keyboard Focus

At runtime, users can move keyboard focus from one control to another in a window or a dialog box by pressing the Tab key. To be included in this sequence, a control's Tab Stop option must be enabled. You enable this option and set the sequence in which controls come into focus by using the Window Editor's Order Controls dialog box.

For step-by-step procedures for setting and clearing a tab stop and for changing the order of tab stops, see [Ordering Keyboard Focus](#) in the CA Aion BRE online help.

Work with Control Groups

Most controls are added directly to a window in the Window Editor. You can, however, create groups of controls without immediately attaching the groups to a particular window. You can then use these control groups on any number of windows.

Create Control Groups

Like standard windows and dialog boxes, control groups are created by subclassing a supplied base class. The default base class for a control group is the GroupBox class in WinLib, but you can choose other base classes, including control groups you create or control groups stored in included custom libraries.

For step-by-step procedures for creating control groups, see [Creating Control Groups](#) in the CA Aion BRE online help.

Attach Control Groups

For step-by-step procedures for attaching control groups, see [Attaching Control Groups](#) in the CA Aion BRE online help.

Radio Buttons

When you add more than one radio button directly to a window, you create a group of radio buttons by default, for all radio buttons that are immediate children of a window interact as a single group unless you divide them into multiple groups.

A group of radio buttons functions as follows:

- The user can either ignore the group (choose none of the radio buttons) or choose one radio button.
- When one radio button in the group is selected and the user chooses another, the first radio button is unselected. (Moving focus to a radio button with the arrow keys chooses the radio button and deselects all others in its group.)
- When focus is on the last radio button in a group and the user presses either the down-arrow or right-arrow key, focus shifts to the first radio button in the same group.
- Similarly, when focus is on the first radio button in a group and the user presses either the up-arrow or left-arrow key, focus shifts to the last radio button in the group.

To create multiple groups of radio buttons on a window, use one of the following three techniques:

- Group the radio buttons with the Group style on the radio button Properties dialog box.

Radio buttons that are set apart from others by the Group style interact with each other as a group, but they do not interact with any other radio buttons on the window.

To create a group of radio buttons with the Group style:

- Make sure the radio buttons that you want to put in a group are listed consecutively in the Order Controls dialog box.
- Assign the Group style to the first radio button in the group.
- Assign the Group style to the next control (of any type) that follows the last radio button in the group.

- Put the radio buttons in group boxes.

Group boxes separate radio buttons visually, not programmatically. Thus, although group boxes help users see which radio buttons are grouped together in a window, you must use group boxes in conjunction with the Group style to create programmatically distinct groups of radio buttons.

To create a group of radio buttons within a group box

- a. Add a group box to the window in the Window Editor, and drag the appropriate radio buttons into it.
 - b. From the Screen menu, choose Order to open the Order Controls dialog box.
 - c. In the Child Windows list, select the group box and move it above the consecutively listed radio buttons that you want to group.
 - d. Since the group box has the Group style set by default, you do not need to assign the Group style to the first radio button in the group.
 - e. Assign the Group style to the next control (of any type) that follows the last radio button in the group.
- Put the radio buttons in control groups, and attach the control groups to the window.

Tab Controls

Tab controls enable you to place multiple groups-or *pages*-of controls in the same area of a window. Building tab controls is a three-step process:

1. Add a tab control to the appropriate window.
2. Add tab pages to the tab control.
3. Add controls to the tab pages.

Descriptions of these steps and detailed procedures are available under the following topics in the CA Aion BRE online help:

- Adding Tab Controls to Windows
- Adding Tab Pages to Tab Controls
- Adding Controls to Tab Pages

Add ActiveX Controls to Your GUI

ActiveX controls—formerly known as OLE controls or OCX controls—are proprietary software objects that you can add to your applications, thereby reusing segments of code written by other developers.

Note: You can add only 32-bit ActiveX controls to your GUI; 16-bit controls are incompatible with Aion BRE applications.

Adding an ActiveX control to your GUI is a two-step process:

1. Add the control to your application.
2. Add the control to your GUI, modifying the control's properties if necessary.

Descriptions of these processes and step-by-step procedures are available under the following topics in the CA Aion BRE online help:

- Adding ActiveX Controls to Applications
- Adding ActiveX Controls to GUI Windows
- Modifying ActiveX Controls
- Inserting OLE Objects into an Application

Merge ActiveX and Application Menu Bars

Some ActiveX controls display menu bars containing commands that apply to the ActiveX control itself.

More Information:

[Persistent Groups](#) (see page 168)

ActiveX Control Properties Dialog Box

If you double-click an ActiveX control or right-click the control and select Properties from the pop-up menu, the ActiveX Control Properties dialog box opens.

Use this dialog to change the attribute name of the ActiveX control as it appears in the parent window class.

To learn more about how to use ActiveX controls in Aion BRE applications, see the following documentation:

- Persistent Groups in the Menu Properties section
- The Generating and Using COM Components chapter in this guide
- The ACTXLib chapter in the CA Aion BRE online help.

Insert OLE Objects into an Application

For step-by-step procedures for adding an OLE object to your application, see Inserting OLE Objects into an Application in the CA Aion BRE online help.

Force Update Modifications from an OLE Object

For step-by-step procedures for embedding an OLE object in your application, see Forcing Update Modifications from an OLE Object in the CA Aion BRE online help.

If the OLE Frame object does not display the changes made to it, you are probably working with an object that does not send regular change notifications to Aion.

Splitter Windows

A splitter window displays information in two panes separated by a vertical or horizontal splitter bar. Users move the splitter bar to resize panes. Each pane can display an instance of any class that:

- Is descended from Window class, and
- Has the WS_CHILD style.

These classes include:

- Image lists
- Tree lists
- Standard windows
- Other splitter windows.

A splitter window must be attached to a container. Valid parents for a splitter window are limited to:

- A standard window
- A dialog box
- A tab page
- Another splitter window

Descriptions of the following processes and step-by-step procedures are available under the following topics in the CA Aion BRE online help:

- Programming Splitter Windows
- Defining Splitter Windows
- Defining Splitter Windows as Vertical or Horizontal
- Creating Splitter Windows Dynamically

Display a Standard Window in a Pane

When a pane displays a standard window, some of the controls contained in the standard window must be resized explicitly. Among these controls are text windows and static text.

To resize these controls, modify the `WhenSized()` event of the standard window. In the method body of `WhenSized()` include code that resizes text windows and static text by calling their `SetRect()` method. Consult the `WhenSized()` method of `FileInfo` class in the Splitter Demo for an example.

Control Properties

A control is made up of two elements:

- A window handle that attaches the control to its parent window (the handle is an attribute of the control's parent class)
- Values that describe the way the control looks and acts on the GUI

Each of these elements has its own set of properties, which you can modify in the following places:

- Use the Properties tab page of the control's Attribute Editor to modify properties that affect the control's role as an attribute of an owning class. These properties include the control's name, its base class, and whether it is private, protected, or public.

To open a control's Attribute Editor, double-click the control's name in the Project Workspace, the Output Window, or the Explorer.

- Use the control's Properties dialog box to modify properties that affect the way the control looks and acts on the GUI. These properties include the control's text, whether it is initially disabled, and its keyboard focus.

To open the Properties dialog box for most controls, double-click the control in its parent window's Window Editor.

The properties contained in the Properties dialog are the topic of this discussion.

More Information:

[Common Object Properties](#) (see page 142)

Common Properties

Common properties have controls in common with other GUI objects,

Auto Validation

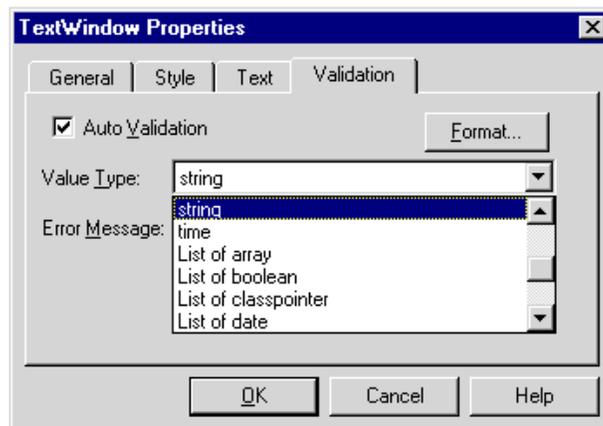
An Auto Validation check box appears on the Properties dialog box of many control types, including check boxes, combo boxes, edit windows, list boxes, radio buttons, and text windows.

If you add a control to an autodialog box or to a dialog box to validate user input and check the control's Auto Validation property, the validation logic that is triggered when a user chooses the OK push button in that dialog box analyzes the control's input to ensure that it is correctly formatted.

Define the Format of User Input

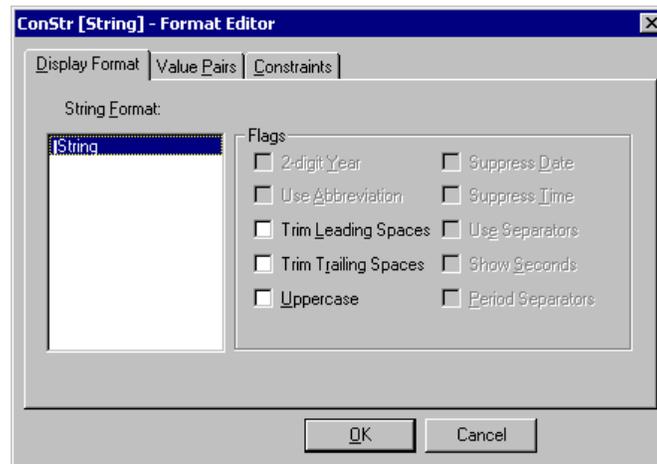
Most controls accept and display only one data type. Check boxes, for example, display or return only Boolean values, represented by checked or unchecked conditions. For this type of control, you indicate that validation logic is to be generated by simply checking the Auto Validation box.

For text windows, however, the Auto Validation feature lets you choose an input format from a number of data type and masking options. When you check a text window's Auto Validation property, the window's Value Type field is enabled:



In this field, you can restrict input accepted by the text window to values of a specific data type. You can then click the Format push button to open the Format Editor and choose a masking option for the selected data type.

For example, the following figure shows the Format Editor for data of type string:



Use the options in the Format Editor to further limit the type of input that the Auto Validation feature permits the user to enter.

Note: The Value Pairs tab of the Format Editor is a data translation mechanism that enables you to link values stored in a database with values displayed or entered on an application's user interface. Use Value Pairs when defining custom data types for an application using the Logic menu's New, Datatype option, not when setting masks for a predefined data type.

Add Menus to Windows

In CA Aion BRE, you add menus to your GUI by creating menu title objects and then attaching them to standard windows (menu titles cannot be attached to dialog boxes). A *menu title* object consists of both the title that is displayed on a window's menu bar and the items that populate the title's drop-down menu. Initially, menu titles are created as independent objects that are not linked to a window. Once created, however, they can be attached to any number of windows.

Create Menu Titles

Like standard windows or dialog boxes, menu titles are created by subclassing a supplied base class. After adding application-specific menu items and logic to this subclass, you can, if you like, use it as the base class for the rest of your application's menus. By doing so, you pass your modifications on to the new menus through inheritance.

In CA Aion BRE, the default base class for a menu title is the MenuTitle class (located in the WinLib library).

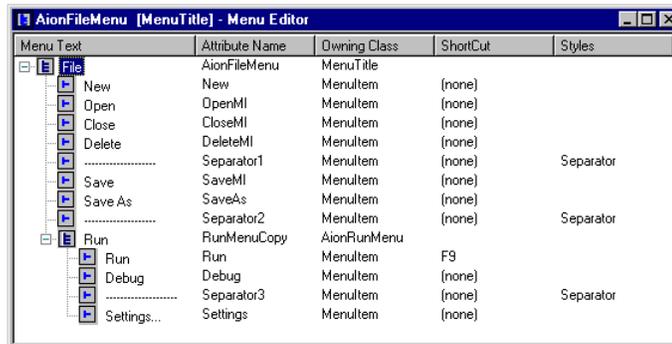
For step-by-step procedures for creating a menu title, see Creating Menu Titles in the CA Aion BRE online help.

More Information:

[Overview of CA Aion BRE Objects](#) (see page 43)

Menu Editor Display

The Menu Editor displays a menu title, its menu items, and their hierarchical relationships to one another. The position of each object in the hierarchy indicates where it appears on the menu at runtime. In the following figure, a menu title called AionFileMenu is open in the Menu Editor. (To display column headings in the Menu Editor, click the pointer in the editor, and then choose Show Labels from the View menu.)



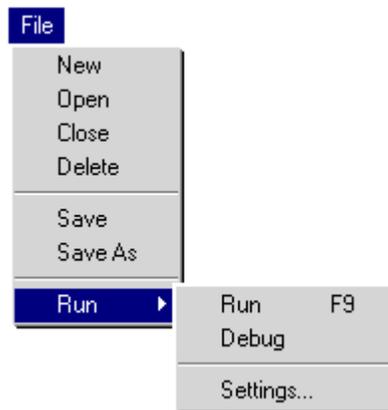
- The object displayed in the first level of the hierarchy under the Menu Text column is the menu title that you are creating or editing (see File in the preceding figure).

If you attach a menu title to a window, the text string in this column is displayed in the window's menu bar.

If you attach a menu title to another menu title (see Run in the preceding figure), you create a submenu, and the text string in this column is displayed in the main menu with a cascading menu symbol (|) to its right.

- Objects in the hierarchy's second level (see New, Open, and so forth, in the preceding figure) are menu items that are displayed when the menu title is chosen and the drop-down menu is opened. Objects in this column can be menu items that execute developer-defined events or menu titles that open cascading menus.
- If there is a menu title in the hierarchy's second level (see Run in the preceding figure), a third level of objects descends from it. These objects are menu items that appear in a submenu.

The following figure shows the runtime version of the menu depicted in the preceding figure:



Add Menu Items to Menu Titles

Once you have created a menu title, use the Menu Editor to create items for the menu title's drop-down menu.

For step-by-step procedures for working with menu items, see the following topics in the CA Aion BRE online help:

- Add Menu Items to Menu Titles
- Rearrange Menu Items
- Deleting Menu Items

Attach Menu Titles to Windows

After creating a menu title in the Menu Editor, you can attach it to more than one window or to another menu title. You can define the logic for the menu title before or after you attach it to a window. If you define the logic before you attach the menu title, the logic is included wherever you attach it. If you define the logic after you attach the menu title to a window, the logic is localized to that window.

For step-by-step procedures for attaching a menu title to a window, see [Attaching Menu Titles to Windows](#) in the CA Aion BRE online help.

Create Submenus

You create submenus-also called cascading menus-by attaching menu titles to other menu titles.

For step-by-step procedures for creating a submenu, see [Creating Submenus](#) in the CA Aion BRE online help.

Pop-Up Menus

You can use menu titles to create not only menus that “drop-down” from a window’s menu bar, but also menus that “pop-up” when users right-click a window or a control.

You create pop-up menus just as you do drop-down menus. Like drop-down menus, pop-up menus can contain submenus. Unlike drop-down menus, however, pop-up menus do not display their top-level titles on the screen.

You can attach pop-up menus to standard windows, to dialog boxes, and to any control derived from WinLib’s Window class.

For step-by-step procedures for attaching a pop-up menu to a window or control, see [Pop-Up Menus](#) in the CA Aion BRE online help.

More Information:

[Create Menu Titles](#) (see page 164)

[Add Menu Items to Menu Titles](#) (see page 165)

Response Invoked by a Right-Click

The Window class in WinLib has a method, `WhenRButtonClicked`, that governs the way applications respond to right-clicks on windows and controls at runtime. Depending on how you implement this method, a right-click invokes the following responses:

- If you specialized the `WhenRButtonClicked` method for the clicked object, CA Aion BRE performs the action specified in the specialized method. (See [To specialize the `WhenRButtonClicked` method for a window or dialog box](#), and [To specialize the `WhenRButtonClicked` method for a control in the Specializing the `WhenRButtonClicked` Method section](#).)
- If you have not specialized the `WhenRButtonClicked` method and ...
 - a. ...the clicked object has a pop-up menu attached to it, CA Aion BRE displays the object's pop-up menu.
 - b. ...the clicked object does not have a pop-up menu but the object's parent window does, Aion displays the parent window's pop-up menu.
 - c. ...neither the clicked object nor its parent window has a pop-up menu, Aion does nothing.

For step-by-step procedures for specializing the `WhenRButtonClicked` method for a window or dialog box, see [Specializing the `WhenRButtonClicked` Method in CA Aion BRE online help](#).

Menu Properties

You can use the `MenuTitle` and `MenuItem` Properties dialog boxes to set property values for a menu title and its individual menu items.

To open the Properties dialog box for menu titles or menu items, double-click the menu title or menu item in the Menu Editor.

Common Properties

The following properties are common to both menu titles and menu items:

Property	Description
Disabled	Grays or dims the menu title or menu item to indicate that users cannot choose it. To enable the title or item, use the <code>SetEnabled</code> method.
Help Text	Displays the specified text in the main frame window's status bar when the menu title or menu item is selected at runtime.

Menu Title Properties

Menu titles also have the following property:

Property	Description
Persistent Group	Controls whether a menu title remains on an Aion BRE application's menu bar when the application's menu is merged with the menu of an ActiveX control.

Persistent Groups

Some ActiveX controls display menu bars containing commands that apply to the ActiveX control itself. To customize the way these menu bars merge with an Aion BRE application's menu bar when an ActiveX control is in focus, use the Persistent Group property in the MenuItem Properties dialog box.

A merged menu can contain up to six groups of menu titles, and each of these groups can contain any number of titles. Aion BRE menu titles can belong only to the first, third, and fifth groups. ActiveX menu titles can belong only to the second, fourth, and sixth groups.

You assign Aion BRE menu titles to groups by entering one of the following numbers in a menu title's Persistent Group field:

- **Zero (0)**-By default, all Aion BRE menu titles belong to Persistent Group 0. Persistent Group 0 menu titles are not persistent and do not appear on merged menu bars. If all the menu titles on an application's menu bar belong to Persistent Group 0, an ActiveX control's menu bar completely replaces the application's when the ActiveX control is in focus.
- **One (1)**-Persistent Group 1 menu titles belong to the first group of a merged menu and thus appear on the leftmost side of a merged menu bar.
- **Two (2)**-Persistent Group 2 menu titles belong to the third group of a merged menu bar, appearing immediately to the right of the first group of ActiveX menu titles (which belong to the second group of a merged menu).
- **Three (3)**-Persistent Group 3 menu titles belong to the fifth group of a merged menu bar, appearing immediately to the right of the second group of ActiveX menu titles (which belong to the fourth group of a merged menu).

Example:

Consider an Aion BRE application that has four menu titles: File, Dates, Settings, and View. When merging takes place, you want to arrange these menu titles as follows:

- Retain File and Dates as the leftmost items on the menu bar.
- Allow the ActiveX control to place menu items to the right of Dates.
- Remove Settings.
- Retain Views and place it to the right of the ActiveX control's first menu title.

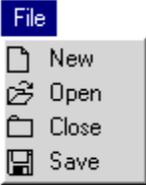
To do this,

- Assign File and Dates to Persistent Group 1.
- Assign the ActiveX control's first menu item to the second group of the merged menu and the rest of its menu items to the fourth group.
- Leave Settings in Persistent Group 0.
- Assign View to Persistent Group 2 (which is the third group of a merged menu).

Menu Item Properties

When you add a menu item to a menu title, the MenuItem Properties dialog opens. In addition to properties common to most objects (see Working with Object Properties), this dialog contains the following properties peculiar to menu items:

Property	Description
Checked	Sets the initial state of the menu item to "checked." By default, a check mark is displayed to the left of checked menu items at runtime. If you want an image other than a check mark to be displayed in this state, specify that image in the Checked Image field of the MenuItem Properties dialog. When a user chooses a checked menu item at runtime, the logic triggered by the user event can call SetChecked(FALSE) to remove the check mark.
Checked Image	Specifies the bitmap displayed to the left of a menu item when the menu item's state is set to "checked." If you leave this field blank, a default check mark is displayed instead. For information about the kind of bitmaps that can be displayed on menu items, see the description of Normal Image.

Property	Description
Normal Image	<p>Specifies the bitmap displayed to the left of a menu item when the menu item's state is "unchecked." If you leave this field blank, no image is displayed in the unchecked state. Each image displayed on the File menu is defined as a Normal Image on the corresponding menu item's Properties dialog.</p>  <p>Specifications for Menu Item Bitmaps</p> <p>Note: For the best results, use monochrome bitmaps. Multicolored bitmaps may create undesirable effects.</p> <p>Note: The space available to display bitmaps on menu items is very small. Since Aion BRE does not automatically resize bitmaps to fit inside this space, you may need to resize bitmaps manually before they will display correctly. For more information about using bitmaps in applications, see Create Resources.</p>
Separator	<p>Displays the menu item as a horizontal bar. Use this style to group menu items visually.</p>

Create Mnemonics and Shortcut Keys

Both mnemonics (for menu titles and menu items) and shortcut keys (for menu items) are set on the Properties dialog. *Mnemonics* are the underlined characters in menu text that are used to open a menu title or to choose a menu item from an open menu. *Shortcut keys* are key combinations that typically include Ctrl, Alt, or Shift. Shortcut keystrokes bypass menus to choose items directly.

For step-by-step procedures for creating mnemonics and shortcut keys, see [Creating Mnemonics and Shortcut Keys](#) in the CA Aion BRE online help.

Menu Conventions

To create user-friendly menus, follow these Windows conventions when building menus for your application:

- Place the File menu on the far left side of the menu bar, and include the Open, Close, and Exit items on File's drop-down menu.
- Place the Edit menu immediately to the right of the File menu.
- Place the Help menu to the right of all other menu titles.
- Use an ellipsis (...) to indicate that choosing a menu item opens a dialog.
Note: Aion BRE automatically uses an arrow (▶) to indicate that choosing a menu item opens a submenu.
- Use separators to group related menu items visually. For example, see the group composed of Save and Save As in the File menu on the Aion BRE menu bar.
- Use mnemonics (one-letter abbreviations) and shortcut keys (keyboard accelerators) to create alternative menu access methods.

More Information:

[Create Mnemonics and Shortcut Keys \(see page 170\)](#)

Add Toolbars to Windows

In CA Aion BRE, you add toolbars to your GUI by creating toolbar objects and then attaching them to windows or dialog boxes. A *toolbar* object consists of both the toolbar "window" and the tool items that the window contains. Initially, toolbars are created as independent objects that are not linked to any window. Once created, they can be attached to any number of windows or dialog boxes.

Create Toolbars

Like standard windows or dialog boxes, you can create toolbars by subclassing a supplied base class. After adding application-specific tools and logic to this subclass, you can use it as the base class for the rest of your application's toolbars. By doing so, you pass your modifications on to the new toolbars through inheritance.

In CA Aion BRE, the default base class for a toolbar is the `ToolBar` class in the `WinLib` library.

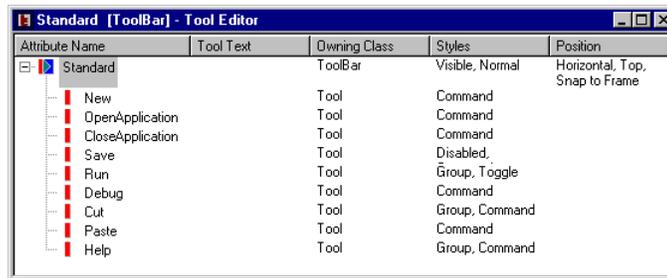
For step-by-step procedures for creating toolbars, see [Creating Toolbars in the CA Aion BRE online help](#).

More Information:

[Overview of CA Aion BRE Objects](#) (see page 43)

Tool Editor Display

The Tool Editor displays a toolbar and the tool items it contains. In the following figure, a toolbar called Standard is open in the Tool Editor:



The toolbar icon at the top of the editor represents a *ToolBar* subclass (see *Standard* in the preceding figure). This subclass contains the tool items listed beneath it in the Tool Editor. To display these tool items in a window, you must attach the toolbar to that window. You can attach more than one toolbar to a window, and you can reuse toolbars.

Descending from the toolbar icon is a list of tool items that the toolbar contains (for example, see *New* in the preceding figure). These tool items are displayed when the toolbar is attached to a window. After creating a *WhenToolItemNameChosen* method for a tool item in the Tool Editor, you can open the Method Editor and write the logic that executes when a user clicks the tool at runtime.

The following figure depicts the runtime version of the Standard toolbar in the preceding figure:



More Information:

[Toolbar Logic](#) (see page 185)

Add Tool Items to Toolbars

Once you have created a toolbar, use the Tool Editor to create tool items for the toolbar. Tool items are the buttons, or *tools* displayed on toolbars that users click to initiate actions in an application.

For step-by-step procedures for performing the following operations, see the CA Aion BRE online help.

- Adding a Tool Item to a Toolbar
- Rearranging Tools
- Deleting Tools

Attach Toolbars to Windows

Once you create a toolbar object (that is, a toolbar and its tool items), you can attach the object to more than one window. In addition, you can define the logic for the object before or after you attach it to a window. If you define the logic before you attach the object, the logic is included wherever you attach the object. If you define the logic after you attach the object to a window, the logic is localized to that window.

Note: You must create a toolbar object in the Tool Editor before you can attach it to a window.

For step-by-step procedures for attaching a toolbar to a window, see Attaching Toolbars to Windows in the CA Aion BRE online help.

Toolbar Properties

You can use the ToolBar and Tool Properties dialogs to set property values for a toolbar and its individual tools.

To open the Properties dialog for toolbars and tool items, double-click the toolbar or tool item in the Tool Editor.

The following properties can be set in the ToolBar Properties dialog:

Property	Description
Enable Balloon Help	Displays descriptive labels, also called "tool tips," under tools when the cursor is placed over them. The text displayed is the text entered in the Help Text field of a tool's Properties dialog box.
Font	Sets the font for toolbars that display text.

Property	Description
Normal	Displays only icons on tools. This is the default style.
Position	Sets the location of the toolbar in the window. Choose between horizontal (top or bottom) and vertical (left or right). Choose Snap to Frame to attach the toolbar to the window's edge.
Show Text on Tools	Displays both icons and text on tools. The text displayed is the text entered in the Text field of a tool's Properties dialog. (To display only text on a tool, leave the tool's Icon field blank.)
Tool Item Size	Specifies the size (in pixels) of tool icons.

Tool Item Properties

The following properties can be set in the Tool Properties dialog:

Property	Description
Group	Adds a space to the left of the tool on the toolbar, creating a visual, but not a programmatically-linked, group of tools.
Help Text	If the Enable Balloon Help style is chosen in the Toolbar Properties dialog, this property provides text for the label that appears beneath the tool when the cursor moves over the tool.
Icon	Specifies an icon to display on the tool.
Tool Item Styles	Command: Creates tools that immediately pop back "up" after a user depresses them.
	Toggle: Creates tools that stay in the "down" position after a user depresses them.
	Exclusive Toggle: Creates a group of Toggle tools in which only one tool can be in the "down" position at a time.

Toolbar Conventions

Tools on a toolbar usually duplicate items in an application's menus but provide quicker access to oft-used commands than the menus do. Since tools can be chosen only with the mouse, it is customary to provide a way to initiate the same action from the keyboard. Typical Windows keyboard alternatives include the following:

- A mnemonic indicated by an underscore in the corresponding menu title or menu item. The user simultaneously presses Alt and the underscored letter.
- A shortcut key displayed next to the corresponding menu item. The shortcut can be any combination of the Alt key, the Ctrl key, the function keys, and alphanumeric characters.

More Information:

[Create Mnemonics and Shortcut Keys](#) (see page 170)

Add Graphics to Windows

Sometimes, a graphic represents a value or a concept more clearly than words or sets of controls do. You can use three types of graphics on standard windows and dialog boxes:

- Bitmaps and icons created outside CA Aion BRE

Many software programs create images that are compatible with Aion BRE applications. Bitmaps used in Aion BRE must conform to the Windows .bmp file format although they do not need a .bmp file extension. Icons used in Aion BRE must conform to the Windows .ico format although they do not need a .ico file extension.

Bitmaps are used to display pictures in many areas of the screen. Icons are often used to build toolbars (for example, see the tool and color bars in the Window Editor).

- Lines, ellipses, and rectangles

Graphics composed of these elements can be drawn directly in the Window Editor.

- By applying these to the preceding graphics described, you can make areas of a graphic respond to the mouse pointer.

For step-by-step procedures for the following operations, see Adding Graphics to Windows in the CA Aion BRE online help.

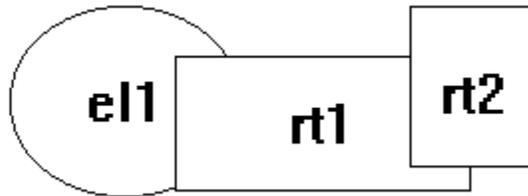
- Adding bitmaps, icons, and graphics
- Adding hot regions to bitmaps

Layered Graphics

Use the Order Controls dialog to layer your graphics on top of each other. The first graphic in the Child Windows list is displayed on the bottom layer of the window; the last graphic in the list is displayed on the top layer of the window.

Example:

Consider the following arrangement of graphics:



In the Order Controls dialog, the graphics in the preceding figure are listed as follows:

- el1, "el1"
- rt1, "rt1"
- rt2, "rt2"

For step-by-step procedures for changing the layering of graphics, see Layering Graphics in the CA Aion BRE online help.

The Order Controls

Use the Order Controls dialog to perform any of the following tasks:

- To rearrange menu items
- To rearrange tools on a toolbar
- To set or clear a tab stop
- To create multiple groups of buttons on a window
- To change the layering of graphics

Create Resources

Graphic images such as bitmaps, cursor shapes, and icons are incorporated into Aion BRE applications as resources. A *resource* is a static instance composed of a handle and a filename. The handle hooks a graphic control to an image file located outside your application. The filename specifies the path to the image file.

You can create resources either as a separate task or as you define a control's properties. All instances of an application's user-created resources are alphabetically listed in the Project Workspace under the Instances node. To open a resource in the Instance Editor, double-click the resource's name in the Project Workspace.

- For step-by-step procedures for either of the following operations, see Creating Resources in the CA Aion BRE online help:
 - Creating a resource without adding it to a control
 - Creating resources while setting a control's property values

Graphics Properties

You can use the Bitmap, Icon, Ellipse, Line, Rectangle, and Hot Region Properties dialogs to set property values for graphics. To open these Properties dialogs, double-click the graphic in the Window Editor.

Each graphic type has its own set of properties. The following table contains a selection of properties drawn from all graphic types:

Property	Description
Auto Drag	Enables users to drag and drop the graphic into a new location. Applies to all graphic types except hot regions.
Data	Specifies the name of the bitmap or icon resource that is displayed on a control. The File button associated with this field opens a standard Windows file dialog box in which you can either choose an existing resource or create a new resource for the control. Note: The Data field must contain the name of a resource's static instance, not the name of the .bmp or .ico file from which the resource was created. To learn more about creating graphic resources for applications, see Create Resources.
Invert Style	Reverses the colors on a monochrome bitmap.
Invert When Selected	Reverses the colors on a monochrome hot region when

Property	Description
	the region is selected.
Pointer	Specifies a mouse pointer for a hot region. When a user moves the pointer over a hot region, the pointer changes to the specified shape.
Text	Specifies descriptive text to display on ellipses and rectangles.

Implement Logic to Run Your GUI

The preceding sections of this chapter explain how to create the classes that you need to construct your GUI. This final section discusses how to instantiate those classes at runtime and how to make them interact with the user and with each other. To do this, you must move your attention from the front end of the GUI to its back end: the application logic.

Application logic is composed of rules and methods that contain instructions governing how an application operates. In graphical applications, most logic is triggered by user actions with the keyboard or mouse. These actions are referred to as *user events*. In Aion BRE, the logic is encapsulated in *methods*, which are small units of code. *Event-triggered methods*, called whenever their related events occur, define an application's response to specific user events.

Events and methods are linked through the methods' names. When a user acts on an interface object, the system looks for a method whose name includes the attribute name of the object and the type of event. For example, when a user chooses a push button named OK, the system looks for a method named `WhenOKChosen` in the subclass of `StandardWindow` or `DialogBox` to which the push button is attached; likewise, when a user chooses a menu item named `FileOpen`, the system looks for a method named `WhenFileOpenChosen` in the subclass of `MenuItem` to which the menu item is attached. If the method exists, it is triggered; if it does not, the system looks for a method named simply `WhenChosen` (without the attribute name) in the control's base class.

Since developer-defined logic in most Aion BRE applications is defined in event-triggered methods and since most event-triggered methods are associated with the controls you add to a window, Aion BRE makes it easy for you to open and write specialized methods through the Menu, Tool, and Window editors.

The following sections discuss the logic associated with particular windows, controls, and graphics.

More Information:

[Write Logic](#) (see page 187)

Window and Dialog Box Logic

An Aion BRE application contains one standard window, called the application window, that opens when the application is started and remains open until the application is closed. All other standard windows and dialog boxes in an application open and close in response to user events. To designate which user events open and close a particular window or dialog box, you must create event-triggered methods that call the predefined `Open`, `OpenModal`, or `Close` methods.

OpenApp, Open, and OpenModal

Each standard window and dialog box comes with predefined methods, inherited from `WinLib`, that can be used to open it:

- The `OpenApp` method opens standard windows only. Use this method to open the *first* window in an application, which is referred to as the application window. After opening the application window, `OpenApp` turns control of the application over to the user.

Only one window can be opened in an application using the `OpenApp` method. This window remains open until the application is closed. Open all other windows with the `Open` method.

- The `Open` method opens both standard windows and dialog boxes nonmodally. Users can shift focus from a *nonmodal* window to another window without closing the nonmodal window. This allows users to work in other windows while the nonmodal window is open.

Use nonmodal windows and dialog boxes for information that does not need to be processed immediately. For example, if users of an order entry application must be able to interrupt the input of a customer return form to take a customer phone order or to open and use another instance of the return form, the windows containing these forms must be nonmodal.

Note: If you send the `Open` method to a window before the application window is open, the window that receives the message does not open.

- The `OpenModal` method opens dialog boxes only. Use this method to open a dialog box modally. To shift focus from a *modal* dialog box to another window in the same application, users must either close or cancel the dialog box. (Users can, however, shift focus to other *applications* on the desktop while a modal dialog box is open.)

Use `OpenModal` when a user response is required before proceeding. For example, a user may be required to enter a login name and a password before other functions can be enabled.

Close

The `Close` method, inherited from `WinLib`, closes windows and dialog boxes and removes them from the screen, passing a return code if a dialog box was opened by a call to `OpenModal`.

WhenOpened, WhenClosed

Sometimes, you need to perform special processing when a window is opened or closed, such as loading data into a Query or sending input to a database. The `WhenOpened` and `WhenClosed` methods let you do this. As supplied, `WhenOpened` and `WhenClosed` do nothing. To use these methods, you must specialize them and add logic.

WhenFocusChanged

Before a user can act directly on a window, he must shift keyboard focus to the window by clicking it. The system indicates which window has focus by highlighting the window's title bar.

Shifting focus from window to window is usually handled automatically by the `WhenFocusChanged` method, which executes when focus moves to or from a particular window. You can, however, specialize this method. For example, Window A contains a list of users. You can edit basic user information in A. Window B contains detailed user information. If changes made in A can affect information in B, you might want B to do a data refresh whenever it gets focus. To accomplish this, specialize the `WhenFocusChanged` method for B by adding refresh code to it.

Remarks

- Opening child window controls

While you must explicitly define methods to open standard windows and dialog boxes, child window controls attached to windows and dialog boxes are automatically opened when their parent window opens.

- Using menu items to open windows

One way to open windows and dialog boxes is by selecting menu items.

For example, when Aion BRE users choose Open from the File menu, a file selection dialog box opens. To do this, the method triggered by choosing Open from the File menu contains only one line of code, which calls the `OpenModal` method to open the appropriate dialog box. After the file selection dialog box opens, a user can either open a file or cancel the dialog box. The logic required to carry out these actions is contained in event-triggered methods created for the Open and Cancel buttons.

- Processing data before opening windows

In some situations, background processing is required before a window or dialog box can open. A menu item called List Employees might contain logic that loads data from a database. After loading the data, the logic calls `Open` or `OpenModal` to open a dialog box containing a report window that displays a list of employees.

- Closing windows versus closing dialog boxes

Most windows are closed by selecting a menu or tool item. Most dialog boxes are closed by clicking a push button. In either case, the method triggered by the user's action must call the `Close` method to achieve the desired result.

- Using push buttons to close windows

Using `Close` and `Cancel` buttons to close dialog boxes requires relatively simple logic. The logic for `OK` buttons is more complex. Before closing a dialog box with the `Close` method, the `WhenOKChosen` method must validate user input in the dialog box. If the input is invalid, the method reports the error to the user without closing the dialog box. If the input is valid, the method takes additional actions, such as writing the input to the database, and then closes the dialog box.

Control Logic

When setting up logic for controls on a window, keep in mind that not every control requires an event-triggered method. Many user events, such as choosing a button or entering text in a window, simply set a value. No logic needs to be defined for these controls.

User events that set values in one control, however, may trigger reactions in or determine the contents of other controls. Frequently, choosing a value affects the availability of other options, and this should be reflected in the window by enabling controls that represent available options and disabling controls that represent unavailable options.

For example, on the Style and Icon tab page of the Aion BRE Tool Properties dialog box, choosing the Group option makes the Exclusive Toggle option available. Hence, the button that represents the Exclusive Toggle option is enabled when the Group check box is checked.

Logic for this kind of control interaction is set up in event-triggered methods. In the preceding example, the logic could be contained in a `WhenGroupChosen` method invoked when the Group option is chosen. `WhenGroupChosen` would use the `SetEnabled` method to enable the Exclusive Toggle option as follows:

```
RBExclusiveToggle.SetEnabled(true)
```

Respond to Multiple Events

There are a variety of user events that can trigger responses in Aion BRE applications, including checking, choosing, collapsing, deleting, editing, expanding, inputting, opening, selecting, and sizing. While many controls respond to only one of these events, some controls—such as image lists, list boxes, menu items, tab controls, and tree lists—can respond to more than one type of user event.

In list boxes, for example, users can single-click an item in the list box to *select* the item, which highlights it but initiates no other action; or they can double-click an item to *choose* it, initiating further processing. To enable list box items to respond to both a single- and a double-click, Aion BRE lets you create a method for each event:

- `WhenListBoxNameSelected`, triggered by a single-click
- `WhenListBoxNameChosen`, triggered by a double-click

The events that a control can respond to are listed in the control's Events dialog box. You define the logic that each event triggers in the Method Editor.

Define Events for Controls

Use the mouse pop-up menu to create methods for controls.

For step-by-step procedures for creating event-triggered methods for a control, see Defining Events for Controls in the CA Aion BRE online help.

More Information:

[Write Logic](#) (see page 187)

Menu Logic

Menu logic is invoked by user events. To make the menus in your application function, define an event-triggered method for each menu *item*. (When a user chooses a menu *title*-which can appear either on a menu bar or, in the case of cascading menus, on a drop-down menu-a drop-down menu opens automatically without the need for developer-defined logic.)

For step-by-step procedures for creating event-triggered methods for a menu item, see Menu Logic in the CA Aion BRE online help.

To learn how to write logic for the method, see the “Writing Logic” chapter.

Typical Menu Logic

The following three sections discuss typical logic implemented in event-triggered methods for menu items.

Opening Dialog Boxes

Choosing a menu item frequently opens a dialog box. To implement this kind of action, the `WhenChosen` method for that menu item must include a call to either the `Open` or `OpenModal` method for the dialog box. An example found in most applications is the `Open` option on the `File` menu. This action can be implemented in a method named `WhenFileOpenChosen`, which needs to contain only one statement:

```
pFileDialog.OpenModal
```

Processing Before Opening a Dialog Box

In some cases, you want to process data before opening a dialog.

Example:

The main window of your application might have a Reports menu title containing a Customer List item. The following logic could be triggered when a user chooses Customer List from the Reports menu:

```
CustomerMarker1 = InputValue1
CustomerMarker2 = InputValue2
Customer_Query.Load
pCustListDlg = CustListDlg.Create
pCustListDlg.OpenModal
```

The first three statements load selected values from a database; the fourth creates the dialog box; and the fifth opens it to display the data in a tree list.

Adding a Check Mark to a Chosen Menu Item

Another common action is to add a check mark to a chosen menu item. This is useful for menu items that toggle between two conditions, such as whether or not a window is displayed (see the Show Output option on the Aion BRE View menu, for example).

To implement this action, make sure the Checked style is enabled in the menu item's Properties dialog. Then have the menu item's WhenChosen method call the predefined SetChecked method.

Example:

If the menu item's attribute name is ShowOutput, the WhenShowOutputChosen method can use the following logic to see whether the menu item is currently chosen and then to display or hide the check mark accordingly:

```
If ShowOutput.GetStyle(MIS_CHECKED) = false
  then ShowOutput.SetChecked(true)
  else ShowOutput.SetChecked(false)
end
```

Toolbar Logic

The toolbar itself does not respond to user input, but each tool item can have a `WhenChosen` method that functions like the `WhenChosen` method for menu items

Tool items can be used to implement methods residing in Aion BRE-supplied libraries. For example, a dialog box with an edit window could have the following functions implemented as tools:

- Cutting and pasting text. The `WhenChosen` methods for the cut and paste tools would call the `EditWindow`'s `CutSelected` and `Paste` methods.
- Finding text. The `WhenChosen` method for the find tool would call the `FindText` method.

For step-by-step procedures for creating event-triggered logic for a tool, see [Toolbar Logic](#) in the CA Aion BRE online help.

More Information:

[Write Logic](#) (see page 187)

[Typical Menu Logic](#) (see page 183)

Graphic Logic

Graphic logic consists of the following:

- Bitmap Response to User Events
- Dragging and Dropping Graphics
- Creating Methods for Graphics

Bitmap Response to User Events

There are two ways to enable bitmaps to respond to user events:

- Create a `WhenBitmapNameChosen` method for the bitmap itself. Users can trigger this method by clicking anywhere on the bitmap.
- Create one or more hot regions and their `WhenHotRegionNameChosen` methods for the bitmap. Use multiple hot regions to associate different logic with different areas of the bitmap.

Dragging and Dropping Graphics

If you assign the Auto Drag style to bitmaps, icons, ellipses, and rectangles, users can move these graphic objects at runtime by holding down the left mouse button and dragging.

To trigger a response when a user releases the graphic after moving it, define a `WhenDropped` method for it.

Create a Methods for Graphics

For step-by-step procedures for creating methods for graphics, see `Graphic Logic` in the CA Aion BRE online help.

Conclusion

Now that you have been introduced to the Aion BRE GUI building tools, you should be able to construct all the windows, dialog boxes, controls, and graphics that your application needs to interact with its users.

If you need more information about particular GUI objects and the classes from which they are derived, see the “WinLib” chapter in the CA Aion BRE online help.

Chapter 6: Write Logic

This chapter provides an overview of Aion BRE methods. It includes examples of common programming tasks where application logic is required. This chapter also contains a discussion of how to use the Method editor and other tools to create application logic.

This section contains the following topics:

- [Write Application Logic](#) (see page 187)
- [About Methods](#) (see page 188)
- [Method Editor](#) (see page 190)
- [Create a New Method](#) (see page 190)
- [How You Program Aion BRE](#) (see page 197)
- [Specialize a Method](#) (see page 206)
- [Write Logic for Windows and Dialogs](#) (see page 207)
- [Process Data](#) (see page 211)
- [Define Other Objects](#) (see page 213)
- [Use the Language Paster](#) (see page 215)

Write Application Logic

Like a procedure or a function, a method contains logic to perform algorithms and calculations. In the body of a method, you access and modify values of attributes and local variables and call other methods. Method logic is written in the proprietary Aion BRE language.

Depending on the functionality you are implementing, Aion BRE application logic can be written from scratch, inherited from a method in a system-supplied library, or automatically generated (as in the case of an event). Among other things, you can use Aion BRE application logic to perform the following runtime tasks:

- Create, open, and close application windows
- Use a dialog box to retrieve user input or show status information
- Control and process data retrieved from a database
- Process Rule logic

Like attributes, methods come in two varieties: instance and class. In this documentation, the un-modified term *method* means *instance method*. An instance method operates directly on the data in an instance. You indicate which instance a method is bound to by using a fully qualified name that specifies the instance or class.

Sample Applications

CA Aion BRE provides many sample applications. To see examples of application logic, open these applications in Aion. You can find the sample application files underneath the Aion BRE installation directory, in the \Examples subdirectory.

About Methods

In CA Aion BRE, all application logic is contained within methods. Methods exist as objects in Aion. Methods can be grouped loosely into two categories:

- **Procedural methods**-You use procedural methods to:
 - Define the high-level flow control logic of your application
 - Set and retrieve values of class and instance attributes
 - Respond to user interface events, such as clicking an OK button
 - Create and delete instances
 - Create, open, and close window instances
 - Access and process data from databases
- **Knowledge methods**-You use knowledge methods to:
 - Define rules
 - Control rule posting and inferencing

Knowledge methods themselves have two forms:

- **Inference method**-A procedural method containing an Infer block.
 - Inference methods may define rules locally, that is, within the Infer block, or invoke a rule method.
- **Rule method**-A non-procedural method containing just the declarative statement of rules.

This chapter focuses on how to write procedural methods. Knowledge methods are discussed in the *CA Aion BRE Rules Guide* and CA Aion BRE Rules online help.

Library Methods

CA Aion BRE provides a set of libraries (such as SysLib, WinLib, and DataLib), referred to as Aion BRE-supplied libraries, or *system libraries*. Methods contained in these system libraries are library methods.

Example:

- **SysLib**-required for all applications; contains an extensive set of methods for manipulating basic data types
- **WinLib**-contains methods for building a user interface
- **DataLib**-contains methods for accessing data stored in a database

Each of these libraries contain many useful methods which can be reused. You can also specialize and inherit them through subclassing. CA Aion BRE supplies other system libraries in addition to these. The CA Aion BRE online help contains detailed information about all system libraries. You can also include custom libraries in an application.

Event-Triggered Methods

Event-triggered methods are those called when an event or action occurs. Most events are triggered from the user interface when a button is pressed or when a menu item is selected. Events can also be triggered when a row is fetched from a database or when an error occurs during a database update attempt. Event method names usually start with "When", as in WhenChosen or WhenFetched.

External Methods

Typically, most methods in an Aion BRE application are written in the Aion BRE method language. However, an Aion BRE application can also call methods whose logic exists in external objects (*external methods*). You might use external methods to reuse code or to access system-level routines not provided by system libraries.

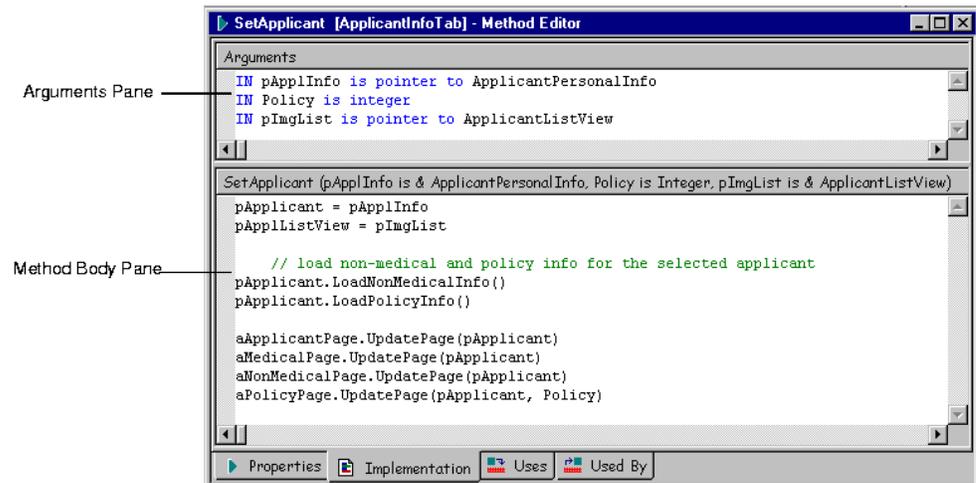
Many of the methods defined in the Aion BRE-supplied libraries are external methods. The DLLs providing the implementation for these methods are located in the directory where CA Aion BRE is installed.

More Information:

[Generate and Use C and C++ Components](#) (see page 335)

Method Editor

Use the Aion BRE Method Editor to write or modify application logic. The Method Editor opens when you create a new method (from the Logic menu, choose New, Method), or when you double-click a method name in the project workspace.



Open an Existing Method in the Editor

In the Project Workspace, methods are indicated by the triangle icon.

For step-by-step procedures for opening an existing method in the editor, see Opening an Existing Method in the Editor in the CA Aion BRE online help.

Note: You can always open the method editor by double-clicking a method name in the Project Workspace, or by double-clicking the method name from within another method's body.

More Information:

[Object Icons](#) (see page 92)

Create a New Method

For step-by-step procedures for creating a new method, see Creating a New Method in the CA Aion BRE online help.

Specify a Method's Properties

View and set a method's properties on the Properties page of the Method Editor.

- The top of the page displays name of the method's *owning class*, which is the class to which this method belongs. Also displayed is the name of the *library* in which the method is located. A method is always located in the same library as its owning class. Both the Owing Class and Library are set when the class is created.

Note: The value in this field updates automatically if you copy or move the method to another class, or if you rename the owning class.

- The Name field contains the name for the method. This is the name used when calling the method.

Note: Aion BRE internally supplies the name for event-triggered methods and method calls. Do not change the name of an event-triggered method.

- Use the Access Type section to determine which other classes and instances can reference the method.

Private

Only the current class or instance can reference the method.

Protected

Only the current class, its derived classes, and instances of those classes can reference the method.

Public

All classes and instances can reference the method.

Note: It is common practice to make attributes private or protected and to provide public methods (*accessor methods*) to get and set the values of the attributes. This practice ensures that other classes and instances cannot directly modify the attribute. Moreover, creating the accessors with the properly formatted names is important for supporting the Java Interface Layer.

For more information related to creator accessor methods, see [Creating Accessors](#) in the CA Aion BRE online help.

- In the Type combo-box, specify the data type of the return value (if there is one).

Note: If a method has a return type, all control paths in the method's implementation must return a value.

The type can be primitive, or a construct (such as list of string, or list of pointer to object).

- In the Arguments field, define any input arguments and output arguments for the method.

Input arguments

If you write a method that operates on values passed to it, you must define an input argument for each value.

Output arguments

These correspond to the values that are passed back to the calling method. Output arguments are also used for values that are passed in to a method, modified, and then passed back to the calling method.

List input and output arguments in the order that they must be specified when the method is called. For each argument, you must specify the data type.

```
IN pAppl is pointer to ApplicantPersonalInfo [= value]
IN policy is integer
IN pRep is pointer to Canvas
OUT str is string
```

value is an optional parameter that specifies a default value for the argument.

Note: It is common practice to list input arguments before output arguments, but it is not required; you can inter-mix input and output arguments.

- In the Style section, choose one or more styles for the method, specifying how the method is defined or how it will be used:

Class Method

When checked, specifies that the method is a class method rather than an instance method.

External

When checked, specifies that the implementation for the method body is external to the application. External methods are typically written in C and must be exported to an entry point in a DLL file accompanying the application. When this box is checked, the External Method Definition tab becomes available.

Disabled

When checked, the method is ignored instead of being used for processing an event. This is useful for methods that are defined in base classes but intended to be specialized in subclasses. You can disable the method in the base class, thereby ensuring that the method is not executed by subclasses that inherit it without specializing it.

Event

This is a system-level property, used internally by Aion. When checked, specifies that the method is a COM event method. The method calls an event method of a COM or OLE object.

DI Member

When checked, specifies that the method is a Domain Interface Member, and the Domain Interface Member Definition tab becomes available.

- External Definition page

If External is checked and you select the External Definition tab, the following fields are enabled:

Convention

Specifies which calling convention to use for the external method. Choose a value from the drop-down list box.

Procedure Name

The name of the routine inside the DLL to execute

Library Name

The name of the DLL containing the called routine

Prototype Button

Click this button to views the prototype for this function. This review can help you ensure that arguments are specified properly.

- Domain Interface Member Definition page

If DI Member is checked and you select the Domain Interface Definition tab, the following fields are enabled:

Type

Click the Condition or Action button to choose the type of DI member.

Label

A symbolic name for the DI member assigned by the application developer

Description

Optional descriptive text associated with the DI member

Comments

Optional comments text associated with the DI member

More Information:

[Return Values](#) (see page 201)

Specify the Method's Implementation

The Implementation page contains the method body (code) as well as any Arguments. You write or modify logic by typing (or pasting) directly in the body pane. Statements can use language operators, data value references, and calls to library methods or application methods.

- The top frame contains the method's arguments. These are the same arguments that are displayed on the Properties page. The arguments are displayed here for easy reference when you are writing the method body. You can edit the arguments from this page or from the Properties page.
- The bottom frame contains the method's body. The method body contains the Aion BRE language statements that are executed when the method is called.

Write the Method Body

Writing a method body can entail:

- Defining local variables
- Choosing the Aion BRE language operators and system-supplied methods to use
- Choosing the custom (user-defined) methods to use
- Ensuring that output arguments and return values are set properly

Edit and Format Features

The Aion BRE Editor pop-up menu gives you rapid access to most common editing operations. Right-click anywhere in the method body pane to display the editing pop-up menu.

- To use Cut, Copy, and Delete, select text before opening the pop-up menu; to use Paste, you must have cut or copied text to the clipboard before opening the pop-up menu.
- To use Shift right, Shift left, Comment, and Un-comment, place the cursor in the appropriate line. These menu items apply to currently selected text or to the line where the cursor is located.
- To affect more than one line, select text on multiple lines before choosing a menu item

The Method Editor toolbar lets you easily search, comment, or format statements.



Note: To specify which toolbars display, from the Aion BRE Tools menu, choose Customize.

Lookup Feature

Use the Lookup feature when writing logic or analyzing an existing method. If, from within method code you double-click the name of an object (method, attribute, constant, and so on), information about the object is displayed.

- Double-click a method name to open it in another Method Editor. Use this feature to look up arguments and return values when writing a method call.
- Double-click any other type of object to open its Properties dialog. Use this when referring to an attribute to look up its access type and its owning class.

You can also use the Lookup option on the Tools menu to search for and display a particular object.

Parse and Save a Method

When the contents of the Method Editor change, an asterisk displays next to the method name in the Editor title bar.

To save the contents of the Method Editor, use toolbar buttons, the menu, or shortcut keys. When you save the method, the asterisk disappears.

Parse the Logic

When the contents of an editor change, the save toolbar is enabled.

The Save Changes button saves the Method Editor to the application, but not to disk. The Method Editor remains open to allow continued editing of the method.

Note: The changes are not saved to disk until you choose Save from the File menu.

In the process of saving the editor contents, Aion parses the method. If a logic error is found, error text displays in the Output pane (at the bottom of the Aion BRE window), and the method icon displays an “invalid” symbol.

The Save Changes button provides an easy way to test object syntax and integrity, as well as perform iterative changes. The Close Editor button closes the Method Editor and prompts to save any changes.

Save the Application and the Editor

To save the application, choose Save from the Aion BRE File menu. If you have not saved changes in the Method Editor, you are prompted to do so.

How You Program Aion BRE

The Aion BRE method language provides the basic constructs for writing complex logic. The language provides conditional control, iteration, comparison, and method and attribute referencing. A comprehensive set of data types and intrinsic functions is provided.

Arguments in Method Calls

When specifying arguments in method calls, match the data type and sequence of the arguments in the called method. Arguments are separated with commas, and can contain literal values (including NULL), expressions, or the name of a value holder.

Note: On UNIX platforms, the maximum number of arguments for an Aion BRE method is 64.

Example:

This specifies a text string as an input argument for a text window instance:

```
NameWindow.SetText ("Type Here")
```

To use an output argument, define a value holder (such as a local variable) in the calling method, pass that value holder to the called method as an output argument, and then set the argument value in the called method body with an assignment (=) statement.

Output arguments are passed by reference, meaning that the called method receives a pointer to the actual data value. Changes made to the data from within the called method affect the calling method.

Example:

This calls a method that returns information in a set of output arguments (which are local variables defined in the calling method).

```
NameWindow.GetFontInfo(facevar,stylevar,widthvar, heightvar)
```

Note: A method call is not required to specify a value or value holder for input arguments that are defined in the called method with an explicitly assigned default value (using = in the definition).

You can pass values into a method with input arguments but you cannot change them in the method body. If you want to modify the value of an argument inside a method, you must use an output argument and set the value of the argument (using an assignment) before the call to the method.

Example:

This is a method named `Build_customer_list` builds a list of customers from a data source.

```
Build_customer_list ("name",cust_list)
```

The input argument determines if the customers are listed by name or company. The logic in the method uses `Add` statements to add the customers to a list named `customer_list`. `Cust_list` is passed back as an output argument. The argument name is a literal input argument that indicates how customers should be listed. `Cust_list` is defined as a local variable in the calling method.

Attribute Data Types

Aion provides a number of predefined data types, such as *string* and *integer*. To indicate that the attribute holds a list of values, use the language construct *list of* (*list of string* or *list of integer*). The system-supplied library called `SysLib` contains the definitions of the data types.

Data type	Description	Example
binary	A binary value can be any arbitrary sequence up to 2 gigabytes. Binary values can be used to build complex structures to pass to external methods.	bitmap image sound video complex structure
Boolean	Holds a value of either true (1) or false (0). The results of comparisons (for example, "If A > B") are Boolean values.	TRUE FALSE
classname	The name of the class whose instances can be values of the attribute.	
constrained data type	The name of a (subtype of a) constrained data type that specifies restrictions on the range of values of the attribute.	
date	Holds any date, including year, month, and day. Use a date data type to create a formatted representation of the date value. Date values can be entered in	June 21, 1998 06/21/1998 21/06/1998 1998/06/21 21-June-1998

Data type	Description	Example
	various formats, as specified by the application.	
integer	Holds a four-byte integer value between -2,147,483,648 and +2,147,483,648. Typically, an integer is represented as a decimal (base 10), octal (base 8), or hexadecimal (base 16) number that represents a value.	1205 +33691 -7898
real	Holds an eight-byte floating-point value between +1.0E+306. Financial data typically requires a real data type.	98.6 -000.02
string	Holds up to 65,512 characters, inclusive. Text data requires a string data type. String values must be surround by quotation marks. To specify quotation marks within a string, use two sets of quotation marks.	"yellow" "Jane Doe" "(800) 555-1212" "Error: ""Not found"""
time	Holds any time value, to the nearest second, for dates after January 1, 1970. Includes the year, month, day, hour, minute, and second. Time values can be entered in various formats, as specified by the application.	June 21, 1998 4:15pm 06/21/1998 4:15pm 21/06/1998 16:15 1998/06/21 16:15 21-Jun-1998 4:15pm
pointer to <i>classname</i>	The name of the class whose instances can be referenced by the value of the attribute.	
classpointer to <i>classname</i>	Holds a pointer to the named class or any class from which that class derives (directly or indirectly). In this way, classpointer allows for an extra level of indirection beyond using a simple pointer to the named class.	

Data type	Description	Example
attributepointer	Holds a pointer to an instance attribute or a class attribute that is not a constant. Attributepointers allow you to refer to another attribute dynamically.	

Local Variables

Local variables are declared within the method body, and are created when control enters the method and released when control exits. These variables cannot be accessed from outside of the defining method. You can use local variables for calculations and looping. A local variable is not an attribute of an instance.

Local variables can be declared dynamically (for example in an IF block). Variables defined this way exist only in the scope of the containing block.

Example:

```
IF a + b then
var i integer
.
.
END
```

In this case, the variable `i` is only recognized inside the IF block, and if referenced outside the block, it is considered an undeclared variable.

The syntax for declaring a variable is:

```
var varname [,varname] [is] data type [= value]
```

where *varname* is the variable name, and the optional *value* is an initial default value for the variable, and which can be a constant or a literal value. *data type* can be a primitive *data type* (such as integer), a construct (such as pointer to or list of primitive data types), or a classname (class containment).

```
var i,j integer
var k integer = 6
var a is string
var pJob is pointer to Jobs
var aPolicy is Policy
```

“pointer to” can also be indicated by prefacing the (pointed-to) object-name with an & symbol.

```
var p &pClass
```

Note: Local variables cannot be declared to be of a type defined by a Constrained Data Type.

Return Values

You can use *return values* to:

- Return a single value. If a method passes only one value to its calling method, you can pass the value using the method's return value. If a method passes more than one value to its calling method, it is common practice to use output arguments for the values.

Or

- Indicate the success or failure of the method. Methods typically return zero (0) for success and nonzero for an error. The nonzero value typically specifies an error code number.

If a method has a return type, all control paths in the method's implementation must return a value. You set the return value with a Return statement inside the method body. The value is then available to the calling method. The calling method does not need to act on the returned value. For example, if a method returns an error code, the calling method can ignore that error.

For step-by-step procedures for defining and using a return value, see Return Values in the CA Aion BRE online help.

Call an Instance Methods

Instance methods operating on a specific instance must specify that instance in the call. The instance name is specified differently if the call is operating on a static instance, such as a connection, or a dynamic instance, such as an interface component.

There are three ways to reference an instance:

- Current
- Name of the instance (static instances only)
- Name of an attribute or local variable whose value is a pointer to an instance

Static Instances

To reference a static instance, use its name.

The following statement calls Commit for a database Connection named local_dBase.

```
local_dBase.Commit
```

Dynamic Instances

Dynamic instances (such as windows or data fetched from a database) are not created until program execution. To reference an instance method for a dynamic instance, use an attribute or local variable that holds a pointer to the instance.

The syntax for calling an instance method is *attributename.methodname*.

Example:

pApp is an attribute of type Pointer to StandardWindow:

```
pApp = AppWindow.Create  
pApp.OpenApp( )
```

Current Instance

For an instance method to operate on the current instance, omit the instance reference from the call, or use the *current* language construct.

You can input current as the default value for an argument. The following code adds the current instance (a specific client) to a list and then saves the list to a database:

```
add(pClientList, current)  
SaveList(pClientList)
```

When an instance method is executing and a method or attribute is referenced *without* being prefixed with an instance reference, the current instance is assumed.

Example:

This example might be used in the WhenOKChosen method of a dialog box to close the dialog. The method body does not need to reference the dialog instance because the current instance *is* the dialog.

```
// Must specify a name
if length(twName.Text) > 0 then
// Close the dialog
  close( )
end
```

Call a Class Methods

Class methods operate on all instances of a class or a selected subset of the instances. In an application with an Employee class, class methods may return values such as the number of employees in a specific department and their average amount of accrued vacation.

For step-by-step procedures for calling a class method, see Calling Class Methods in the CA Aion BRE online help.

Associations

To maintain an association between instances of two classes, it often advantageous to define the association as a subclass of SysLib's `_Associations` class. Subclasses of `_Associations` are known as association classes. The Association editor allows the programmer to create association classes.

There are two advantages of defining an association as an association class:

- You can assign properties to the association. For example, `MarriageDate` may be assigned to the association class `Marriage`.
- You may use the roles specified in the Association editor as public attributes (instance pointer or lists of instance pointer depending on the multiplicity of the association) of the associated classes. Using roles is especially helpful if the associations change membership during execution of the knowledge base.

To establish an association between two instances of the associated classes, you should call the `Establish()` methods of your association class and pass it pointers to instances.

Example:

```
var pMarriage is pointer to Marriage
pMarriage = Marriage.Establish(phusband, pwife).
```

`Establish()` creates an instance of the Marriage association and returns a pointer to that instance. This instance maintains the pointers to each of the associated instances. Related instances are represented by an instance of the association class. To delete the association, invoke the association instance's `Dissolve()` method.

Example:

```
pMarriage.Dissolve( ).
```

Note: If the call to `Establish()` violates a constraint on the association, Aion will abend. To address this, it is recommended that the association's `OkToEstablish()` method be called before calling `Establish()`. `OkToEstablish()` returns a Boolean that allows the programmer to proceed with the establishment of the association or handle the error condition.

The use of roles is especially helpful. For example, if `Owns` designates the role of `PropertyOwner` to `Property`, the following construction yields the properties owned by a property owner:

```
pInstanceOfPropertyOwner.Owns => List of Property instances
```

Aion BRE automatically maintains this list. If the property owner sells a property, that is, if a particular instance of the Ownership relationship is dissolved, the next time the preceding statement is invoked, the sold property will not be part of the `Owns` list.

Similarly, the owner of a given property can be referenced through the `OwnedBy` role (defined as the role of `PropertyOwner` in the Ownership association. For example:

```
pInstanceOfProperty.OwnedBy => pInstanceOfPropertyOwner
```

Even though roles function as public attributes of the associated classes, they do not appear as attributes on these classes.

More Information:

[Association Editor](#) (see page 104)

Attribute and Class Pointers

Besides pointers to instances, Aion BRE provides two additional types of pointers: attribute pointers and class pointers. This section describes how to assign values and use these new types of pointers.

Attribute Pointer

An attribute pointer points to an attribute. Attribute pointers are typically used when specifying the goal attribute in a chaining statement and in the meta-programming capabilities of Aion BRE.

Attribute pointers are usually assigned by means of system methods, for example, `Pointer::LookupAttribute()`. To assign a value directly to an attribute pointer you must use the `"->"` operator:

- For instance attributes, the syntax is `->pointer to instance.Attrib1`. The pointer to the instance can be omitted when the attribute pointer refers to an attribute of the current instance. This omission commonly occurs when specifying the goal attribute in a chaining statement.
- For class attributes, the syntax is `->classpointer.Attrib1`. Where `Attrib1` is a directly accessible class attribute, `classpointer` may be omitted. (A directly accessible class attribute is a class attribute of the current class or any unique public class attribute.)

Class Pointer

A class pointer holds a pointer to the specified class or to any of that class's subclasses. Thus, polymorphism is supported through class pointers just as it is through instance pointers.

In general, you can specify a class pointer using one of two alternative syntaxes:

- The standalone class name, for example, `BusinessInfo`.
Exception: When declaring an initial value of an attribute defined as a classpointer, the `"->"` operator is required.
- The `"->"` operator, for example, `->BusinessInfo`.
Exception: When dereferencing a classpointer, the stand-alone class name must be used. An expression of the form `->BusinessInfo.Attrib1`, where `Attrib1` is a class attribute, is construed by the interpreter to be an attribute pointer to `Attrib1`. Expression of the form `->BusinessInfo.Method1()` causes a syntax error.

The following groups of expressions are equivalent:

```
var pcl is classpointer to BusinessInfo
pcl = BusinessInfo
pcl = ->BusinessInfo
and
GetDerivedClasses(BusinessInfo)
GetDerivedClasses(->BusinessInfo)
```

Only class attributes and class methods can be accessed through a class pointer. The most common use of a class pointer occurs when you qualify the name of a class method or attribute with the name of the owning class. For example, when you create an instance of a class, you must qualify the `Create()` with the name of class whose `Create()` method you wish to invoke:

```
MyClass.Create()
```

In this case, "MyClass" is, in fact, a class pointer.

Specialize a Method

Specialization is the process of distinguishing class objects from the parent objects from which they were derived.

You can specialize any available method, including those supplied with Aion BRE. You might specialize a library method to modify actions built into the system. Or, you can specialize an event method to add behavior to the default actions of a window or control.

When you specialize a method, the default logic for the method is still available. You can perform the default logic any time during the execution of the method by calling the method defined in the parent class. The methods you are most likely to specialize are `WhenOpened` and `WhenClosed` from `WinLib`, and `WhenFetched` from `DataLib`.

Specialized versions of these methods include up to call a method using the parent class of the current instance:

```
return(up.WhenUpdated(argument))
```

Note: If you are working with a developer-created class library, or creating a library of your own, you may find yourself specializing methods other than those supplied with Aion BRE.

For step-by-step procedures for specializing a method, see [Specializing a Method](#) in the CA Aion BRE online help.

Unspecialize a Specialized Method

You can unspecialize a method that has previously been specialized. Unspecializing deletes the specialized logic and returns the method to its default function.

Note: When you unspecialize a method, any specialized code in the method is deleted.

For step-by-step procedures for unspecializing a specialized method, see Unspecializing a Specialized Method in the CA Aion BRE online help.

Write Logic for Windows and Dialogs

In a graphical application, you write logic to create, open, and close application windows and dialog boxes. Typically, windows are opened or closed in response to user events (for example, when the end-user chooses a menu option or clicks a button).

Note: CA Aion BRE supplies prewritten methods for many common tasks, such as Closing and Opening windows. For these functions, you do not need to write methods from scratch; you simply access these prewritten methods from your code, or subclass and specialize the corresponding Aion BRE-supplied class.

Create and Open the Application Window

By default, an application window is opened when an Aion BRE application begins executing. While, at edit-time you use the Window Editor to *define* the look of this initial window, the window is *created* and opened at runtime.

Typically the logic for creating and opening this initial window is located in the Start method of the entry class. When you first create an Aion BRE application (using the New option from the File menu), Aion automatically generates the logic needed to create and open the initial window, and places the logic in the Start Method of the Main class.

The default (generated) Start method uses the following code to create and open the application window:

```
pApp = AppWindow.Create( )  
pApp.OpenApp( )
```

A Create method is called, and returns a pointer to the initial window. You can then use the pointer to reference the window in Open(), Close(), or other window method calls.

Create Dialogs at Runtime

Typically, an application has several dialogs for gathering user input and displaying information. You can choose to create all dialogs at one time in the entry class, or to create them upon demand.

For a small number of dialogs, the easiest strategy is to create all dialogs in the Start method of the entry class, store pointers to them in public class attributes defined in the Main class, and then open and close the dialogs throughout the application.

This code fragment creates two additional dialogs, a List Dialog and a Dept dialog, where pListDlg and pDeptDlg are class attributes defined in Main:

```
// Create application dialogs
pListDlg = ListDlg.Create( )
pDeptDlg = DeptDlg.Create( )

// Create and open application window
pApp = AppWindow.Create( )
pApp.OpenApp( )
```

However, as the number of dialogs grows, this strategy can impose a heavy burden during program initialization.

An extension to this strategy is to delay window creation until the dialogs are actually needed. In this case, you do not use the Start method to create dialogs; you use the method that opens each dialog to create it. The following code fragment demonstrates this technique:

```
// Create the dialog
if (pListDlg = NULL) then
    pListDlg = ListDlg.Create( )
end

// Open it
pListDlg.OpenModal( )
```

Note: All class attributes have an initial value of NULL.

The DialogBox Class

Dialog Boxes are used at runtime to gather input from a user or to show status information. In CA Aion BRE, a dialog is represented as an instance of the DialogBox class in WinLib. The attributes of a Dialog Box contain information about it, and there is an attribute for each control contained within the dialog. You can open dialogs by calling either the OpenModal method or the Open method. Calling OpenModal opens a modal dialog at runtime, whereas calling Open causes a modeless dialog to open at runtime.

Set Initial Values for Controls

Before opening a dialog, you must set its initial control values. You can specialize the Aion BRE WhenOpened method to set these initial values, thereby encapsulating related logic in the dialog.

Each control provides public methods to set its value. For example, you can set the contents of a text window using the SetText method

Example:

```
pListDlg.twName.SetText("Jones")
```

You can set the contents of a list box using the SetStrings method

Example:

```
reps = list("Fred", "Sally", "Bill")  
pListDlg.twReps.SetStrings(reps)
```

You can specify the initial values of many controls within the Window editor; however, in some cases the initial state of the controls depends on the state of other attributes. These controls must be initialized programmatically just before the call to OpenModal or Open:

```
// Set initial values  
pListDlg.twName.SetText("Jones")  
pListDlg.rbMarriedStatus.SetChosen(FALSE)  
  
// Open the dialog  
pListDlg.OpenModal( )
```

Use Dialogs to Get User Input

User input is obtained from a control on the dialog. Typically, the input is gathered from an event method defined for a push button.

Each control that can accept user input has an attribute that stores its current value. When accessing the value of a control, you refer directly to the public attribute of the control that holds the value.

Example:

A radio button has a Boolean attribute named `Chosen` that stores whether the button has been selected. This attribute can be referenced in your method as follows:

```
if rbMarriedStatus.Chosen = TRUE then
    ...
end
```

The preceding example does not prefix the reference to the attribute (`rbMarriedStatus`) with the pointer to the dialog (`pListDlg`). This is because the dialog instance is the *current instance* and therefore the attribute reference is assumed to be a part of the dialog. In general, writing instance methods that work with the current instance provides the greatest reusability of objects.

Attributes of other control types may have a different name and data type, but you handle them in the same general way. With a text window, for example, the value set by the user is stored in the string attribute named `Text`

Example:

```
if twName.Text = "Smith" then
    ...
end
```

Report Status Using a Modeless Dialog

Using a modeless status dialog is a common approach to reporting status to the user during long operations (such as a percentage-completed indicator). A dialog of this type has a single text control.

The dialog is opened just before the long operation begins, and then at specific points during the process, status is written to the text window.

```
// Create and open status dialog
if (pStatusDlg = NULL) then
    pStatusDlg = StatusDlg.Create( )
end
pStatusDlg.Open( )

// Perform long operation
for idx = 1 to 100000
    CalculateSomething( )

    // Update status every 100 calculations
    if (idx mod 100 = 0) then
        msg = format(idx) & " calculations performed"
        pStatusDlg.twMsg.SetText(msg)
        pStatusDlg.Refresh( )
    end
end

// Close status dialog
pStatusDlg.Close( )
```

Process Data

Another common task for logic involves working with rows of data retrieved from a database.

Iterate Instances of a Query

Typically, after data is loaded as instances, you iterate the instances to perform application-specific logic. The following code block shows a method that loads all employees of a company, then builds a list of strings used to set the values of a list box where names is a local variable of type list of string and pEmp is a local variable of type pointer to EmployeeQuery.

```
var names list of string
var pEmp &EmployeeQuery

// Load the data
EmployeeQuery.Load( )

// Iterate over the instances of the class
names = NULL
for EmployeeQuery, pEmp
    // Add the name to the string list
    add(names,pEmp->EMP_NAME)
end

// Set the names into the list box
pListDlg.lbNames.SetStrings(names)

// Delete the query instances
EmployeeQuery.Flush( )
```

Use Markers to Control Data Selection

Using markers is the most common way to control data selection from a database. It is better to use markers to control the select statement of a query than to create many different queries. The following example sets the values of three markers based on the values of the controls on a dialog. After the marker values are established, the query is loaded.

```
// Get the criteria from the user
if (pListDlg.OpenModal( ) = 0) then
    return
end

// Set representative markers
mkRep = pListDlg.coRep.selection
if (length(mkRep) = 0) then
    mkRep = NULL
end
```

```
// Set contact marker
mkContact = pListDlg.twContact.text
if (length(mkContact) = 0) then
    mkContact = NULL
end

// Set problem type marker
mkType = NULL
if pListDlg.rbBug.chosen then add(mkType,"SFTWR")
if pListDlg.rbEnh.chosen then add(mkType,"ENHRQ")
if pListDlg.rbDoc.chosen then add(mkType,"DCERR")

// Load the data
ProblemQuery.Load( )
```

Define Other Objects

You can also define the following three additional types of attributes. Create these objects using their respective editors (from the Logic menu, choose New, *objtype* where *objtype* is the type of object).

- Attributes
- Classes
- Instances

Note: When creating a new object, remember to specify the class where it is defined. By default, the owning class is based on the current selection in the Project Workspace, or the owning class of the current method.

Attribute Editor

Use the Attribute Editor to define an attribute. From the Logic menu, choose New, Attribute to open the editor.

Attributes

Attributes are value holders that declare storage in every instance of the class where they are defined. For example, an employee name might be an attribute defined in an Employee class.

Class Attributes

Class attributes are value holders that declare storage within the class where they are defined. Additional storage is not declared in the instances of the class. Therefore, class attributes are used to declare global information for the class itself. For example, the average salary for all employees might be stored as a class attribute of the Employee class.

To define a class attribute, check the Class Attribute check box on the Attribute editor.

Constants

Constants are value holders stored within the class where they are defined. The value of a constant is established during development, and you cannot change it during program execution. You cannot specialize constants.

To define a constant, check the Constant check box on the Attribute Editor for a Class Attribute.

Accessor Methods

To automatically generate Accessor Methods (that is, Get and Set methods), when an attribute is selected, use the Create Accessor option from the Logic menu.

More Information:

Creating accessor methods, see [Creating Accessors](#) in the CA Aion BRE online help.

Instance Values Dialog

Instances are the actual objects of an application. Usually you create them dynamically, but you can also create them statically during program development. Static instances are useful when an application has predefined data values for attributes unlikely to change during execution.

One example of static instances is found in the Connection class in DataLib. Aion creates a static instance of the Connection class when you choose New Connection from the Data menu, using the values you supply to set the values of the Interface, Data Source UserID, and Password attributes.

Use the Language Paster

Use the Language Paster to insert a predefined language statement of one of the following kinds into the Method editor:

Operators

Logical operators such as equal to (=) and less than (<)

Constructs

Logical language structures such as If/Then/Else

Rules

Code that executes knowledge

For step-by-step procedures for using the Language Paster, see Using the Language Paster in the CA Aion BRE online help.

Chapter 7: Access Data

This chapter examines accessing data from Aion BRE applications, including establishing a database connection, defining queries and stored procedures, loading data, and committing transactions. In addition, it describes the way in which CA Aion BRE accesses IBM MQSeries functionality, and supports the use of MQSeries-compliant objects.

This section provides an overview of the process whereby CA Aion BRE accesses and manipulates data from server databases.

This section contains the following topics:

- [Data and Aion BRE Classes](#) (see page 217)
- [Define a Database Connection](#) (see page 222)
- [Define a Query](#) (see page 222)
- [Write SQL Statements](#) (see page 231)
- [Define a Stored Procedure](#) (see page 232)
- [Data Test Facility](#) (see page 236)
- [Load Data from a Database](#) (see page 237)
- [Save Modifications to the Database](#) (see page 243)
- [Database Errors](#) (see page 248)
- [Define Records and Serialize Data](#) (see page 249)
- [MQLib to Access MQSeries](#) (see page 253)

Data and Aion BRE Classes

Data is loaded into CA Aion BRE in the form of instances of a class. The application-defined class is a subclass of either the Query or StoredProcedure class (these are predefined in DataLib).

- Use the Query class if the data is the result of an SQL SELECT statement.
- Use the StoredProcedure class if the data is the result of a stored procedure invocation.

When an SQL SELECT statement or a stored procedure is issued against the database, the database retrieves all rows that satisfy the request and returns the set of rows, or the *result set*, to the application. When data loads into an Aion BRE application, each row of the result set is transformed into a runtime instance of the class that initiated the database interaction.

Data Manipulation

Once the data is loaded in the form of instances, the application can manipulate the data visually or programmatically.

- Visual display and modification of data is controlled using the windows and controls provided by WinLib and IOWLib.
- Programmatic manipulation of the data is accomplished using the Aion BRE language to change attribute values.

Update the Database

DataLib tracks all visual or programmatic modification of query instances, such as modifications made to attribute values, or the creation or deletion of an instance. Then, DataLib automatically generates the necessary INSERT, DELETE, and UPDATE SQL statements to reflect the changes back to the database.

To ensure integrity, when data is accessed by several users at the same time, DataLib provides two modes for transaction management.

- Default-commits updates to the database after every UPDATE, INSERT, or DELETE SQL statement
- Application-defined-updates during a transaction are committed to the database as a unit

Basic Steps in Working with Data

The following basic steps enable Aion BRE to access and manipulate data from a server database:

1. Define a database connection.
2. Create a query or stored procedure invocation to specify the SQL SELECT or EXECUTE statement and the structure of the result set.
3. Write logic to load data into the query, or use the Auto-Load feature.
4. Write logic to manipulate data programmatically, and logic to save modifications to the database.

Define a Database Connection

Establish a connection to a data source, such as an ODBC data source, or to a database such as Oracle or Sybase. The connection, once established, takes the form of a static instance of the Aion BRE Connection class. An application can create and use any number of connections.

Note: To access a database through ODBC, a data source must have been defined using the ODBC Administrator. To access a database using a native driver, you must know the server machine's network name or address (or its alias).

Invoke a Query or Stored Procedure

A Query defines the following:

- SQL SELECT statement used to obtain a result set
- Field attributes that map to the columns in the result set
- Connection on which the database interaction takes place

You can define any number of queries for a connection. Queries are defined using the Query Editor.

Write the SQL Statement (Query Editor)

Use the Query Editor to write the SQL statement that initiates database interaction. The Query Editor has a built-in SQL parser to verify the basic syntax of the SELECT statement. The SELECT statement can include a class attribute defined in the Query, or a *marker*. The value of the marker is substituted in the SELECT statement during execution.

The Query Editor includes a graphical display of catalog information so you can choose which columns participate in the SELECT list of the SQL SELECT statement.

Create the SELECT List

The SELECT list specifies which columns to retrieve from the database when the SELECT statement is issued. You can use any number of columns from one or many tables to establish the SELECT list.

Note: CA Aion BRE also supports multiple table updates.

For every column chosen from the catalog information pane, the Query class creates a field attribute to hold the value of the column when data is loaded.

Test the Query

Once you define the field attributes, markers, and a SELECT statement, you can test whether the query properly executes against the database. The Test option on the Data menu (available when the Query Editor is open) runs the SELECT statement against the database, and then displays returned data in tabular format.

More Information:

[Define a Query](#) (see page 222)

Write Data-Loading Logic

Once components are defined, you write logic to initiate the database interaction. The only required statement is the invocation of the Load() method (of the Query class).

Invoking the Query class Load() method causes all rows of the result set to be loaded into the application as runtime instances of the Query class.

Example:

The following example shows a typical code block establishing values for two markers, and then calls the Load method to retrieve matching rows from the database.

```
// Set the WHERE clause markers
mkDepartment = "Finance"
mkLocation = 177

// Load all matching employees
MyQuery.Load( )
```

Note: The preceding example assumes you are Auto-Loading (which is the default). If you choose to Manually Load rows, the code is slightly different.

Write Logic to Update the Database

Once data has been loaded, DataLib automatically begins tracking modifications. You can modify data using Aion BRE language statements. DataLib tracks changes made to attribute values, newly created instances, and instances that are deleted.

To update the database with data changes, use the Save() method. Save() issues the appropriate SQL INSERT, DELETE, or UPDATE statement for each modification outstanding against the Query data.

Example:

The following code fragment shows an example of data loading, followed by modification that results in both DELETE and UPDATE SQL statements being issued when the Save() method is called.

```
// Set the WHERE clause marker
mkDepartment = "FINANCE"

// Load all matching employees
MyQuery.Load( )

// Loop through the instances of the Query
var p is pointer to MyQuery
for MyQuery, p

// Delete all temporary employees
    if p.Status = "TEMP" then
        p.Delete( )

// Give everyone else a raise
    else
        p.Salary = p.Salary + 100
    end
end

// Update the database
MyQuery.Save( )
```

Transaction Management

Connections control transactions with the database. By default, DataLib defines a transaction as a single SQL UPDATE, INSERT, or DELETE statement. Therefore, during a Query's Save() operation, each SQL statement is committed individually.

By using the Connect, Commit, and Rollback methods of the connection class, DataLib lets an application define its own user-defined transaction.

More Information:

[Update Data with Automatic and Manual Commit Modes](#) (see page 244)

Define a Database Connection

A *connection* is a conduit between a database and an Aion BRE application. When appropriate methods are invoked during program execution, data flows from the database into the application or from the application back to the database.

CA Aion BRE uses a database driver to communicate with the database. Currently, DataLib supports the following native drivers:

- Oracle 10g and 11g
- Sybase 12.5.4
- Microsoft SQLServer 2005 and 2008
- IBM DB2 for LUW 8.1 and 9.1
- IBM DB2 for zOS 8.1 and 9.1
- ODBC (for accessing a wide range of databases and files)

Note: To assure that DB2 database access is successful, the user must use the same version of DB2 Client and Server.

To work with database data (both at design-time and runtime), at least one connection must be defined. The attributes of the Connection class hold information necessary for making the physical connection to the database. For example, connecting to a Sybase database using the native Sybase driver requires a user name, password, location of the host machine, and sometimes a database name; all of which is stored in the Connection class.

For step-by-step procedures for defining a connection in your application, see *Creating a Connection* in the CA Aion BRE online help.

Define a Query

The Query class defines a read/write database interaction initiated by an SQL SELECT statement. The Query class uses Field Attributes to map to columns in a result set, and Class Attributes to hold information about the database interaction, such as the following:

- The SELECT statement
- Any marker values within the SELECT statement

When your application calls methods at runtime, data you describe in the Query Definition window (of the Query Editor) is loaded into the application. After data is loaded, DataLib tracks all modifications to instances of the Query so it can automatically update the database with appropriate changes.

A query definition is developed using the Query editor.

Each definition contains the following components:

- One Field Attribute for each column in the result set

Note: Each Field Attribute is mapped to a field in the database.

- Any number of additional attributes to hold application-defined information. (Additional attributes are often used to hold values calculated from values of Field Attributes)
- Specialized Class Attributes that define properties of the query, such as the SELECT statement, and a pointer to the Connection instance
- Any number of additional Class Attributes to define variables for the SELECT statement

Note: A Class Attribute used as a variable in the SELECT statement is called a Marker.

- Inherited methods that provide the ability to load from a database, and save modifications back to the database
- Any number of application-defined methods

Use Inheritance to Reuse Queries

Most queries are derived directly from the Query class. It is possible, however, for any query to serve as the base class for a new Query. The new query inherits the Field Attributes and properties of the base class. Typically, both the result set and the SQL SELECT statement are further specialized.

Because all field attributes from the base Query are inherited and cannot be removed, the SELECT list of the derived Query's SELECT statement must contain the original columns in the original order.

New field attributes can be defined on the derived Query, but because locally defined field attributes are ordered after inherited field attributes, you need to specify new field attributes at the end of the SELECT list.

Note: Use the %columns, %tables, and %markers directives in the SELECT statement to automatically generate the SELECT list from field attribute definitions.

Concurrency Control

Concurrency control refers to the Aion BRE object locking options. You choose a locking option at the query level to apply to all transactions processed through the query.

CA Aion BRE uses optimistic locking. To enforce read-only locking, verify that none of the field attributes are specified as Key. To enforce pessimistic locking, add the corresponding keywords to the SQL statement.

Create a Query

The New Query dialog provides necessary fields for defining a query and starts the Query Editor.

For step-by-step procedures, see [Creating a Query](#) in the CA Aion BRE online help.

Query Editor

Use the Query Editor to define a query for an Aion BRE application. An application can contain an unlimited number of Queries.

Use the Query Editor to:

- Display catalog information associated with the connection
- Easily create field attributes and map them to columns in the result set
- Create field attributes and map them to calculated fields in the SELECT list of the SELECT statement
- Create markers for the SELECT statement
- Use the SQL Paster to assist in writing the SELECT statement
- Test the query against the database.

Open the Query Editor

To open the Query Editor, right-click on a Query Class and choose Open from the pop-up menu.

The Catalog Information pane graphically displays the available tables and columns on the connection associated with the query. Use the Catalog Information tree to browse tables within the database and to browse the columns defined for each table.

Note: If, when the connection was defined, filters were removed for other object types (such as synonyms), those types appear in the Catalog Information pane as well.

The Query Definition pane contains all Attributes and Markers defined for the query. As you create Field Attributes from columns, they display in this pane.

The SELECT Statement pane contains the SELECT statement. By default, it appears as above, with placeholders for markers and attributes. You can also hard-code a SELECT statement here.

View Table Data

For step-by-step procedures for viewing table data, see Viewing Table Data in the CA Aion BRE online help.

Include Columns

For step-by-step procedures for including one or more columns in a query, see Including Columns in a Query in the CA Aion BRE online help.

Change Field Attributes

You edit field attributes in an Attribute Editor, which features tables to display field and data properties. Double-click on a field attribute in the Query Editor to display the attribute in the Attribute editor.

More Information:

[Field Attributes](#) (see page 226)

Write the SELECT Statement

The Select Statement pane (in the lower-right corner of the Query Editor) contains the SELECT statement for the query. Following is the default SELECT statement.

```
select %columns from %tables where %markers
```

It specifies that all field attributes defined for the query will participate in the SELECT list. DataLib automatically builds the SELECT list at program execution.

You can add to this statement by typing directly in the field, by pasting from the clipboard, or by using the SQL Paster

More Information:

[SQL Paster Utility](#) (see page 232)

Test the Query

Use the Data.Test facility to test whether a query has been properly defined. This facility takes the query's SELECT statement, the list of field attributes, and the initial values of any defined markers, and constructs the SELECT statement that will execute during program execution.

To test the query, choose Test from the Data menu in the Query Editor.

Field Attributes

Use the Field Attribute Properties dialog to identify key fields, to mark fields as unmapped, to inherit from your predefined data types, and, when necessary, to modify the system-generated data type for the field.

The Field Attribute Properties dialog is accessible whenever the Query Editor is open.

For step-by-step procedures for creating and modifying field attributes, see [Creating Field Attributes in the CA Aion BRE online help](#).

Calculated Fields

Field attributes can be used to display values that do not directly correspond to database columns. Use the following guidelines with calculated fields.

- If the field is calculated by application logic, set up the field as Unmapped (meaning the field is not involved in database interaction). You can then use application logic to assign a value to the field. (Frequently, the value is calculated by the WhenFetched method.)
- If the field is calculated by an SQL expression, write the SQL expression in the alias for the field you are adding.

For step-by-step procedures for adding a calculated field, see [Adding a Calculated Field in the CA Aion BRE online help](#).

Use Markers with Query Classes

A marker is a class attribute defined within the Query class you are defining. You can use markers as variables, particularly within the WHERE clause of the SELECT statement.

Mapped and Unmapped Markers

Mapped markers have an alias property that specifies table.column. Mapped markers are used only with the SQL directive %markers. Use mapped markers when writing a WHERE clause with many AND expressions. The default SELECT statement uses mapped markers to search for all non-NULL columns. The SELECT statement is automatically expanded for markers that are lists.

```
select %columns from %tables where %markers
```

If you wish to eliminate a clause from the list of mapped markers, assign the mapped marker the value NULL.

Note: The value NULL in Aion BRE is not the same as reserved word NULL in SQL. You cannot use mapped markers to query columns whose value is NULL. You must either hard-code the condition, for example, WHERE %markers AND (Age IS NULL), or use unmapped markers.

For step-by-step procedures for creating a mapped marker, see [Creating a Mapped Marker](#) in the CA Aion BRE online help.

Use unmapped markers to assign a value to a specific query column or when using an OR clause. Assign a value to a marker through application logic, just as you would assign a value to another attribute. The following SELECT statement uses unmapped markers to specify an employee's name and department.

```
select %columns from %tables
  where EMP_NAME = :mkEmpName and
        DEPT_ID = :mkDeptId
```

When an unmapped marker is set to NULL in Aion BRE, the entire contents of the clause within parentheses containing the unmapped marker is replaced by `1=1`, which always yields *True*. This feature supports *dynamic clause elimination*. Dynamic clause elimination allows you to include or exclude a condition in the SQL WHERE clause at runtime. For example, to dynamically eliminate the clause (Age IS NULL) create an unmapped marker of type Boolean. In the SQL query statement specify (where mkAgeNull is the name of the Boolean unmapped marker)

```
...WHERE...AND (1=:mkAgeNull AND Age IS NULL)
```

Setting mkAgeNull to TRUE will cause the SQL clause to be rendered (1=1 AND Age IS NULL); setting mkAgeNull to NULL, will cause the clause to be rendered simply as (1=1).

Important! In order to avoid inadvertently eliminating other conditions of the query, bound the unmapped marker portion of the query conditions with parentheses, even if those parentheses would not otherwise clarify or determine the meaning of the overall search clause.

Note: To map an unmapped marker, double click the name of the marker in the Query editor and add an alias in the Marker Properties dialog.

For step-by-step procedures for creating an unmapped marker for a Query class, see [Creating a Marker](#) in the CA Aion BRE online help.

For step-by-step procedures for modifying a query, see [Modifying Queries](#) in the CA Aion BRE online help.

Change the Properties of a Query

The Query Properties dialog allows the programmer to expand the information retrieved by the query editor from the database (by default, the query editor retrieves table and views), and to specify how Aion BRE handles features of the query. For example, you can use this dialog to specify whether the extended names convention is used in the database, which requires quoting of names in the query; how NULL values will be treated when they are brought into CA Aion BRE; and whether static SQL will be used to perform the query.

Important! If Use Static SQL is checked, the query class name becomes restricted to a length of 18 characters. This restriction--which must be considered when creating the query class--is imposed by the pre-compiler that generates the C file from the .sqc file.

More Information:

[Dynamic versus Static SQL](#) (see page 228)

Dynamic versus Static SQL

SQL statements can be invoked by Aion BRE query classes either statically or dynamically. The default state of query class is to execute SQL statements dynamically. You can use the Query Properties dialog to set individual query classes to use static SQL.

Note: Currently static SQL is only available for DB2 on Windows and MVS platforms.

Dynamic SQL

- The dynamic SQL statements are constructed and prepared at run time. With dynamic SQL you can perform the following functions not available with static SQL:
- Dynamic SQL statements can be changed at runtime.
- Query class attribute marker values can be NULL with dynamic SQL.

Note: Fully qualified table names (owner.table) must be used when creating a dynamic connection to a DB2 database.

Static SQL

The source form of static SQL statements generated for Aion BRE applications are written in the C language. Static SQL statements are prepared before the program is executed, and the operational form of the statement persists beyond the execution of the program. For any given static SQL query, the following SQL statements are generated:

Note: Currently NULL values for markers are not supported with static SQL.

- SQL Select
- Update
- Insert
- Delete

Note: The query must specify a key field and have at least one field designated read/write for these statements to be generated. If there is no key field specified the update, insert, and delete statements will not be generated.

The steps in the preparation of static SQL statements are:

Pre-compilation

The pre-compilation of the static SQL statements is performed by respawn. This process checks the syntax of the SQL statements and, if successful, generates a bind file and a pre-compiled C file.

Binding

During the binding process, the bind plan generated during the pre-compile step is bound to the database.

Compilation and Linking

During this process, the pre-compiled C file generated during the pre-compile step is compiled and linked into a library.

Build Queries Using Static SQL

This section describes the requirements and the process for building queries using static SQL. When you build an application with Static SQL, Aion BRE generates the standard JCL, plus an additional file containing the static SQL statements (.sqc file). The build process for Static SQL is invoked via respawn.exe. Before starting RESPAWN, you must enter the RESPAWN command in a command window creating using the DB2 command DB2CMD.

Note: Aion BRE applications can not be built with static SQL through the Aion BRE IDE.

System Configuration Requirements

You must have DB2 client v2.1.2 with SDK or higher installed on your system.

Start RESPAWN

Note: Before starting the RESPAWN process, all File and Library settings must be configured through the IDE.

For step-by-step procedures for starting RESPAWN, see Starting RESPAWN in the CA Aion BRE online help.

More Information:

[Run and Build Applications](#) (see page 439)

RESPAWN Static SQL Build Process

To RESPAWN initiates the static SQL build process, follow these steps:

1. Source files are generated.
Aion generates an .SQC file for static SQL.
2. Aion compiles the application.
3. Aion invokes mscsdb2.bat, which completes the following processes:
DB2 prep (pre-compilation)-The following steps are included in the DB2 prep:
 - Connect to DB2, invoking the pre-compiler
 - Generate C file and Bind filesDB2 bind (binding)-The following steps are included in the DB2 bind
 - Static SQL statements are bound
 - Disconnect

4. Linking of the .OBJ files is completed.
5. Clear of all associated files and processes.

Note: RESPAWN compiles only one application at a time. If you have included libraries, ensure that RESPAWN is executed for each user-defined library. This process is not essential for system libraries.

Write SQL Statements

After defining Field Attributes in a query, the application contains a mapping of the database tables and columns it needs to access. To make specific selections from the tables and columns, provide an SQL SELECT statement.

The default SELECT statement automatically selects all columns from all tables referenced by field attributes. To define a more complex selection criterion, modify the default SELECT statement.

For example, add a WHERE clause to constrain the records returned in the result set. Use an ORDER BY statement to control ordering of rows in the result set.

The SELECT statement is the only SQL statement you define in an application. The SQL UPDATE, INSERT, and DELETE statements are generated automatically by DataLib when the Save method is invoked.

Note: If required, direct execution of non-SELECT statements can be issued directly against the connection (using the Execute() method of the Connection class).

If you wish to modify (or hard-code) the statement, you can type directly in the Select statement pane, paste from the clipboard, or use the SQL Paster utility.

SQL Paster Utility

Use the SQL Paster to build a SELECT statement by selecting from lists of SQL constructs, catalog information, and relevant application components (such as field attributes and markers).

To open the SQL Paster from the Query Editor, choose SQL Paster from the Data menu. The SQL Paster dialog displays.

Note: Before using the SQL Paster, verify the cursor is at the correct paste location in the SELECT statement.

For step-by-step procedures for setting up static WHERE, GROUP BY, and ORDER BY clauses, see the following topics in the CA Aion BRE online help:

- Setting Up a Static WHERE Clause
- Setting Up a GROUP BY Clause
- Setting Up an ORDER BY Clause

Define a Stored Procedure

A *stored procedure* is a combination of SQL statements and flow-control constructs used to perform database functions such as data updates and queries.

Stored procedures are created (and reside) on the database server. Typically, they are written and maintained by a database administrator; however, many stored procedures are provided directly by the database vendor. Stored procedures can accept input arguments, and, for certain databases, can return a result set.

- An Aion BRE application can invoke a stored procedure when a program is running. Stored procedures offer the following advantages:
- They execute more rapidly than SQL statements residing in your application.
- They have a centralized location.

Also, if modifications to a stored procedure are necessary, you do not need to modify, recompile, and redistribute your application (unless, of course, the stored procedure name or its arguments change).

Queries and Stored Procedures

During program execution, DataLib treats a stored procedure invocation like a read-only query. Stored procedures differ significantly from queries.

Unlike queries, stored procedures have the following properties:

- They do not support the Save method. Any updates to the database are completely encapsulated within the stored procedure on the database.
- They use the SQL statement EXECUTE instead of SELECT. Therefore, their flexibility is limited by implementation on the database.
- They have widely varied implementation support from database to database. For example, few stored procedure implementations support returning a result set. Also, syntax for invoking a stored procedure is not standard among databases.
- They have limited catalog support from database vendors. As a result, manual definition of the result set and arguments of a stored procedure is often required.

Create Stored Procedures

The New Stored Procedures dialog provides necessary fields for defining a stored procedure and starts the Stored Procedure Editor.

For step-by-step procedures, see [Creating Stored Procedures](#) in the CA Aion BRE online help.

The Stored Procedure Editor

The Stored Procedure editor is an environment in which you can construct a Stored Procedure class.

Use it to:

- Display catalog information associated with the connection.
- Create field attributes and map them to columns in the result set.
- Create markers for the arguments of the Execute statement.
- Test the stored procedure against the database.

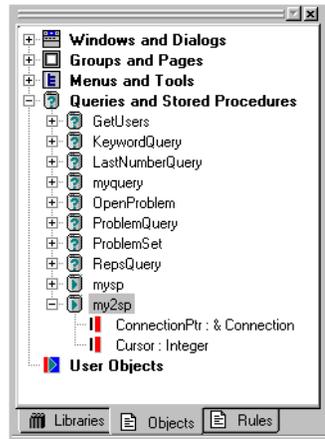
Catalog Information Pane

The Catalog Information pane graphically displays all stored procedures available on the connection (specified when you created the Stored Procedure). You can use the Catalog Information Tree to browse the stored procedures defined on the database, and, if support for arguments is available from the database driver, the arguments defined for each stored procedure.

The catalog information is displayed hierarchically; to expand or contract it, double-click on the stored procedure name. When a stored procedure is expanded, its arguments display. In addition, if the database provides the necessary support, the result set columns display.

Locating Stored Procedures in the Application

Use the Objects tab of the Application viewer to view all stored procedures and queries in the application.



For step-by-step procedures for reviewing or modifying a stored procedure, see [Reviewing or Modifying Stored Procedures](#) in the CA Aion BRE online help.

Define the Result Set

If the stored procedure returns a result set, all columns in the result set associated with the stored procedure are returned. You cannot specify a subset of columns to return.

For step-by-step procedures for defining the result set, see [Defining the Result Set](#) in the CA Aion BRE online help.

Add Markers to Stored Procedures

A marker is a class attribute defined within a StoredProcedure class. You assign a value to a marker through application logic, just as you assign a value to any other attribute.

You can use markers as variables within the EXECUTE statement. Typically, markers hold the values for input arguments to the stored procedure. Use markers to receive return values and output argument values from the stored procedure.

Note: For output arguments, specify the OUTPUT keyword in the EXECUTE statement.

Example

This example uses markers to specify the name and department of an employee for a stored procedure on a Sybase database.

```
exec get_employee @name = :mkEmpName,
                 @dept = :mkDeptId
```

Example:

This example uses markers for receiving an integer return code and sending a message string from an Oracle stored procedure:

```
exec :mkRetCode get_message (:mkMsgStr)
```

For step-by-step procedures for creating a marker for a stored procedure, see *Creating a Marker* in the CA Aion BRE online help.

Writing the EXECUTE Statement

The lower right pane of the Stored Procedure Editor is the Execute Statement pane, used to write the EXECUTE statement for the stored procedure. The syntax of the EXECUTE statement varies according to the conventions of the database accessed.

For information about the syntax requirements of a particular database, consult the appropriate database documentation or ODBC documentation. The following table lists the request syntax for each type of database driver. Text in square brackets [] represent optional arguments:

Database	Request Syntax
ODBC	call procedure_name [(arg1, arg2, . . . argn)]

Database	Request Syntax
	Examples: call get_employee call get_employee ('SMITH', NULL, 'SALES')
SQL Server and Sybase	exec procedure_name [arg1, arg2, . . . argn] Parentheses are not used. Examples: exec get_employee exec get_employee @name = 'SMITH' exec get_employee 'SMITH', NULL, 'SALES'
Oracle	exec [return_arg] procedure_name [(arg1, arg2, . . . argn)] Examples: exec get_employee exec get_employee (name => 'SMITH') exec get_employee ('SMITH', NULL, 'SALES') exec :mkReturn get_code

Test the Stored Procedure

Use the Data.Test facility to verify that the Stored Procedure is properly defined. Data.Test takes the EXECUTE statement of the Stored Procedure, the list of field attributes, and the initial values of all defined markers, and constructs the EXECUTE statement that executes when the program is run.

Data Test Facility

The Data Test facility tests queries and stored procedures during development.

To open the data.test facility, choose Test from the Data menu when the Query Editor or Stored Procedure Editor is open.

The Data Test dialog displays. It contains the fields described in the following sections.

Select Statement Field

The Select Statement field displays the SELECT statement from the query after all marker substitutions have been performed. To make changes to the SELECT statement, use the Query editor, and then restart the Data.Test facility.

Markers List

The Markers pane lists the markers defined for the query. Double-click a marker name to see its properties (including initial value). Data.Test substitutes the marker's initial value in the SELECT statement.

To change the initial value of the marker, in the list box, double-click on a marker to display the Marker Properties dialog.

When finished, you can click Execute to re-execute the SELECT statement after modifying it or the marker's initial values.

Result Set Field

The Result Set pane shows the results of issuing the SELECT statement against the database. When you invoke Data.Test, the SELECT statement automatically executes, and, if successful, displays the results in this field.

To control the maximum number of rows retrieved, use the Maximum Number of Rows to be Fetched control (at the top of the dialog).

Load Data from a Database

CA Aion BRE can load (retrieve) data in Automatic mode or Manual mode. Automatic mode is the default.

The following sections apply to the Automatic loading mode.

Load Data

You can set up a data loading process using the Automatic mode very easily. Once you have a query defined, write a simple Load statement to load the result set of the query into Aion BRE instances.

In Automatic loading mode, when you call the query Load method, all rows selected by the query definition are loaded. The query manages the database cursor.

By default, multiple rows are fetched at a time during database load processing.

The number of rows that are fetched per database request is established by the ROWSET_SIZE attribute of the Data class.

When the ROWSET_SIZE is one, a single row fetch is used instead.

The performance of a multiple row fetch is estimated two times faster than a single row fetch.

Note: You will rarely need to alter the ROWSET_SIZE attribute.

Cursor Management

Unlike Manual mode, when you load data using Automatic mode, the query manages the database cursor. You need not write additional logic to control opening and closing of the database cursor.

In Automatic mode, a Load call is not preceded by a Connect call. To combine manual commit with Automatic load, use a Connect call to initiate manual commit mode:

```
Server_connection.Connect(true)
    //initiates manual commit mode
Inventory_query.Load
    //loads data in default mode
```

where *Server_connection* is a connection and *Inventory_query* is a query class.

Note: If you define markers for a query, assign values to the markers before loading the data.

Manual Load Mode

Loading data in Manual mode gives you more flexibility than Automatic mode because you control the database cursor and load rows individually with calls to the Fetch method. With Manual loading mode, you explicitly write the logic to establish the connection. Further explicit calls open the database cursor, fetch the desired data, and finally, close the database cursor.

Loading data manually means creating more method calls. This in turn means more flexibility, because you can evaluate each row as it is fetched and decide whether or not to display it. You can also calculate an additional value based on the data and display it.

Database Cursor Processing

Whenever you establish a connection to a database, you open a cursor in the database. In Manual mode, you write the logic that contains the step-by-step processing of the database cursor you establish with the Open method call. The step-by-step logic used in manual load mode is made up of four method calls for a specific query:

Connect

Manually establishes the database connection

Open

Initializes the database cursor

Fetch

Fetches and loads data into database columns

Close

Disengages the database cursor

The Open operation opens the related connection and opens a cursor in the database. Call Fetch to load rows, and end with a Close call. You can write a WhenFetched method to analyze or decode each row as it is loaded. WhenFetched executes as part of the load operation. The steps you take after loading data depend on how you are handling the data. The next section demonstrates different ways to handle the Open-Fetch-Close calls.

Manual Loading Examples

In Manual loading mode, you open a cursor in the database, fetch rows, and then close the cursor. Opening the cursor opens the appropriate connection, as in the following:

```
Inventory_query.Open  
Inventory_query.Fetch  
Inventory_query.Close  
//manual load and default commit modes
```

To operate in both Manual load and Manual commit modes, use the following:

```
Server_connection.Connect(true)  
Inventory_query.Open  
Inventory_query.Fetch  
Inventory_query.Close  
//manual load and manual commit modes
```

Manual Cursor-Processing Examples

The following example is from an application containing these components:

- Customer_Connection-Connection that references a customer data source
- Customer_Query-A query with a SELECT statement
- Cust_List_Dialog-An autodialog with three text windows for displaying CustomerName, CustomerID, and CustomerAddress

To load the data, write the Connect, Open, Fetch, and Close statements. Try placing the Fetch in a Loop construct as in the following example:

```
//Set markers for loading data
CustID1 = IDWindow1.Text
CustID2 = IDWindow2.Text

//Open the connection
Customer_Connection.Connect
Customer_Query.Open

//Loop getting data
Loop
    //Break on end of data
    if Customer_Query.Fetch = 100 then
        break
    end
end

//Close database
Customer_Query.Close
```

Note: The code block includes setting the values for initializing a pair of markers.

WhenFetched()

The data selection criteria provided by SQL SELECT statements is sometimes insufficient. Your application may require more refined data selection to evaluate rows of data as they are loaded into an application, or refined save and commit logic when you are saving instances back to the database.

The WhenFetched method provides additional control and power to the data loading process. The WhenFetched method is executed for every row that is retrieved using the Fetch and Load methods. The WhenFetched method is executed whenever a row is fetched from the database. It processes each row of the result set (determined by the query SELECT statement) as it is retrieved.

An Aion BRE developer can create a WhenFetched method for any query or stored procedure.

The WhenFetched method also lets you accept (using TRUE) or reject (using FALSE) each row of the result set as it is loaded. You can process each row returned from the database to determine whether to create a corresponding instance in Aion BRE. WhenFetched operates like other When methods. The method is invoked automatically once for every row returned.

Enable WhenFetched()

The WhenFetched method defined in DataLib is disabled. Specialize the WhenFetched method and enable it in the Query class for which WhenFetched processing occurs.

Use WhenFetched()

The WhenFetched method is an instance method. Each time a row of the result set is read from either the Load or Fetch statement of a Query or from the execute processing of a Stored Procedure, CA Aion BRE engages the WhenFetched method.

This means data from the current row of the result set is loaded into an Aion BRE instance, and the WhenFetched method is activated on that instance. Aion BRE architecture lets you examine the columns of the selected row-by referring to the attributes of the current instance from within the method. Once the WhenFetched method has control, your logic begins executing. The full power of the Aion BRE language lets you control processing.

At some point, the method you create must issue a return statement and a value of either TRUE or FALSE. If the method returns FALSE, the instance is deleted. If the method returns TRUE, the data values are accepted and processing continues with the next row. Typically most rows are accepted.

In Automatic loading mode (the default), WhenFetched is called during the Load operation. In Manual loading mode, WhenFetched is called by Fetch.

You can use WhenFetched to find a specific column's value (for example, maximum or minimum), and store it in a class attribute. This lets you avoid returning the value in every row of the result set. Consider the following query, which queries the salary of every employee as well as the maximum salary. The name of the maximum salary column is Changes.

```
SELECT NAME,SALARY,MAX(SALARY) FROM EMPLOYEE
```

This SQL statement returns one row in a result set for every employee. Each row has three columns. The third column, MAX(SALARY), is identical for every row.

You may prefer to issue the following SQL statement and use it in conjunction with a WhenFetched method:

```
SELECT NAME,SALARY from EMPLOYEE
```

In the method body of the WhenFetched method for the preceding query shown, the following logic is used to find the maximum value in the Changes column:

```
If max_salary < current.salary
then
max_salary = current.salary
end
return(true)
```

WhenFetched methods always return a Boolean value of true or false. A return value of TRUE indicates that the row is accepted and loaded as an instance of the class into the Query. A return value of FALSE indicates that the row is not loaded.

You can extend the WhenFetched method (used in conjunction with the previous method shown) to compute the average or standard salary deviation.

```
num_employee = num_employee + 1
avg_salary = avg_salary +
  (current.salary - avg_salary) /
  num_employee
```

You also can exclude from the average salary computation any row containing undesired data.

Example:

To exclude any salary equal to 0 (as may exist in the case of an employee who is paid hourly) in computing the average.

```
if current.salary <> 0 then
  num_employee = num_employee + 1
  avg_salary = avg_salary +
    (current.salary - avg_salary) /
    num_employee
end
```

Save Modifications to the Database

This section discusses the various ways of saving modifications you make to the database.

Data-Update Mechanisms

Typically, updating data is achieved by writing code that generates the Save method. Once loaded, data can be updated in these ways:

- User input into components of your application's graphical interface
- Language statements issued by application logic

Update Data with Automatic and Manual Commit Modes

The call sequence for saving data depends on the commit (transaction management) mode you are using. There are two commit modes for creating SQL inserts, updates, and deletes, and for managing transactions: Automatic commit mode and Manual commit mode.

- Automatic commit mode is the default mode. In Automatic commit mode, calling Save (for the query that loaded the data) generates SQL statements (UPDATE, INSERT, and DELETE) and sends them to the database. The format of the call is straightforward:

```
Inventory_query.Save
```

- In Manual commit mode, calling Save for the query generates the same UPDATE, INSERT, and DELETE statements, but they are held in the connection until you call Commit or Rollback. This lets you perform verification or checking before committing the transactions. The basic sequence of method calls is Save followed by Commit or Rollback.

```
Inventory_query.Save  
Server_connection.Commit
```

Use Manual mode to check after saving and to determine whether to commit or roll back the database transaction.

Note: The form of the Connect statement determines whether you will use Manual commit mode.

More Information:

[Manual Commit Mode](#) (see page 245)

Automatic Commit Mode

Unless you indicate otherwise, your application runs in Automatic commit mode. The query tracks all changes to loaded data, and holds the current data values as a single operation.

To commit data, call the query `Save` method. `Save` automatically generates SQL `UPDATE`, `INSERT`, and `DELETE` statements, and sends them to the connection. The connection's role in Automatic mode is to commit each statement as it is received. There is no `Rollback` method in Automatic mode.

To initiate Automatic Mode Processing, call `Save` for the query, as in the following example:

```
Customer_Query.Save
```

To clear rather than save changes, call `Flush` for the query, as in the following example:

```
Customer_Query.Flush
```

Note: Flushing cancels pending inserts, deletes, and updates. It also clears the query, freeing up space taken by unneeded data, and ensures that database information retrieved by subsequent queries is not duplicated.

Manual Commit Mode

In Manual commit mode, the query tracks all changes to loaded data and holds the current values, but as separate operations. When you call `Save`, the query generates SQL `UPDATE`, `INSERT`, or `DELETE` statements, and sends them to the connection.

The connection does nothing until you call its `Commit` method. Then it commits all pending SQL statements. You can still flush the query, clearing any pending updates. You also have a `Rollback` method that lets you roll back the last set of database updates pending SQL statements.

Enable Manual Commit

To use manual commit mode, call the `Connect` method with an argument of `TRUE`. This disables Automatic commit mode and lets you set up logic for Manual commit mode.

Example:

```
Customer_Connection.Connect(TRUE)
Customer_Query.Save
//perform any status checking required
Customer_Connection.Commit
```

The query generated SQL `UPDATE`, `INSERT`, and `DELETE` statements for all data changes in the query when `Save` was called. These statements are held by the connection, so you can perform any status checking required by your application. Depending on the outcome of your status check, you can call `Commit` or `Rollback` for the connection.

Important! If there is already have an active connection (initiated by the `Load` or `Loadlist` methods), transitioning from automatic mode to manual mode requires that a `disconnect` is called before the manual mode connect.

WhenUpdated()

As has been previously stated in this chapter, the data selection criteria of SQL `SELECT` statements may be limited. Your application may require a more refined data selection to evaluate rows of data as they are loaded into an application, or refined save and commit logic when saving instances to the data source.

Use the `WhenUpdated` method to transform data as it is returned from the application back to the database. `WhenUpdated` provides more specific data-saving criteria than `INSERT`, `UPDATE`, or `DELETE` operations triggered by the `Save` method.

Multiple Table Queries and Outer Joins

Queries that select columns from multiple tables also support updates across all included tables. However, each included table must have at least one field attribute defined as `Key`.

You can use the field attribute *Outer Join* style in conjunction with `WhenUpdated` to insert rather than update a row in a table during `Save` processing.

Example:

For example, if you had a table of problems and another table of optional problem characteristics, a query might use an outer join to manipulate the problem and its characteristics as one unit. If the problem did not have an entry in the characteristics table, NULL is returned for all columns. However, if characteristics are assigned to a problem that did not have any previously, the update process needs to insert a row, rather than update one.

Use the following syntax:

```
//Update the value for the outer join field attribute
if (joinfield_table2 = NULL) then
    joinfield_table2 = joinfield_table1
end
```

When to Use WhenUpdated()

Include WhenUpdated routines to evaluate updates generated by Save. A WhenUpdated method for a query or stored procedure is executed every time an INSERT, UPDATE, or DELETE is processed. The function of the WhenUpdated routines is to accept or reject each instance. According to the criteria defined in the method, the save is accepted or rejected.

You can specialize and enable the WhenUpdated method, process the arguments described, and use it to process SQL statements (UPDATE, INSERT, and DELETE) generated during a Save operation. You can use the SQL_* constant to determine which kind of statement is being processed, and take appropriate action. Specialized versions for WhenUpdated must send *Up* to accept the row for an update. Use the following syntax:

```
return(up.WhenUpdated(argument))
```

In the preceding example, the argument could be SQL_DELETE, SQL_INSERT or SQL_UPDATE.

To reject the update, return(FALSE).

Example:

For example, use `WhenUpdated` to check for a valid salary value before updating a row to an employee database, as in the following example (which returns a Boolean value). A return value of `TRUE` indicates that the update is accepted. A return value of `FALSE` indicates that the update is rejected:

```
//Check for salary and reject if invalid
if(current.salary < 0)
    Then return(FALSE)
end
//Update row for proper salary
return(up.WhenUpdated(nSQL_op))
```

Database Errors

Errors can occur when you use an application to access and manipulate data in an external database. In Aion BRE, possible data access errors can occur during the following operations:

- Connecting to the database
- Loading (fetching) data from the database
- Saving data back to the database

Other types of errors may occur during `Fetch` and `Save` operations. Aion BRE provides the following three methods for reporting database errors:

GetError

Called by `WhenError`, retrieves any native database errors issued. The errors are returned to `WhenError`, which reports them to the user

WhenError

Reports on the Aion BRE database interaction interrupted by the error. It also calls `GetError` to retrieve any native database errors issued

WhenSaveError

Processes any errors that occur during a `Save` operation

Runtime errors that are caught in a runtime library (such as `DataLib`) must be reported by the library writer. Because `DataLib` sends `WhenError` to the connection on which the error occurred, the error reporting can be customized by users of the library.

Define Records and Serialize Data

Records are typically defined by a Cobol Data Division 01 level, a struct in C, or a DSECT in OS/390 Assembler. The values of attributes of an object can be mapped into a record structure. To aid Aion BRE programmers in defining records and performing record I/O, Aion provides ReLib. Record I/O is defined as the need to write out multiple attributes (of mixed types) as one continuous piece of storage, and to read one continuous piece of storage into multiple attributes.

ReLib provides the ability to **serialize** a record as a binary stream, that is, to write a complex structure as one stream and then to restructure that stream into the proper format by parsing its serialized form. Record I/O is accomplished by “flattening” a series of attributes into one binary buffer. The reverse process is accomplished by parsing multiple attributes out of one binary buffer. Serialization enables the Aion BRE application to write and retrieve the attributes of an object as a binary stream. (Object serialization is an important feature of the Java language.)

ReLib supports complex record structures such as records that consist of other records. For more information on ReLib, see the “ReLib” chapter in the CA Aion BRE online help.

Note: Physical I/O of a record is performed using existing Aion BRE capabilities.

Construct Records

A record is defined by instantiating the Record class and populating its Elements attribute (defined as a list of pointers to Element) with pointers to the constituent instances of the Fields class and/or other Records making up the record's structure. Fields are represented by instances of the specific *DatatypeField* classes that hold the values for the fields of the record. Care must be taken to list the pointers to the elements of the record in the appropriate order that represents the structure of the record.

The following sections describe the attributes of the Field and Record classes.

Field Attributes

Field supports the following datatypes (see the **type** attribute / TYPE_ constants):

- Binary
- Integer
- Real
- String
- Variable length string
- Pointer to a string value

Note: that the other Aion BRE datatypes such as Date and Time may be built from other types, byte by byte, by using a Binary Field.

The following Notes pertain to Field attributes:

- All datatypes except VarStringField and PointerStrField have their **length** attribute fixed at edit time. These methods have a SetLength() method. (No accommodation is made for calculating string lengths based on a trailing NULL.)
- The **usage** attribute specifies whether a Field is used as input-only, output-only, or filler Fields according to the following constants:

USAGE_FLATTEN_ONLY:

Field will not be parsed; that is, its space in the Binary buffer will be skipped over without writing to any attribute.

USAGE_PARSE_ONLY:

Field will leave the corresponding space in the Buffer untouched during flattening.

USAGE_FLATTEN_AND_PARSE (the default):

Field is to be both written to and read out of the buffer.

USAGE_FILLER:

Field is filler only. USAGE_FILLER suppresses writing to the buffer and reading the buffer to an attribute.

Aion provides static instances Filler1 through Filler7 defined as USAGE_FILLER, which are provided as a convenient way to specify alignment.

Work with DatatypeFields

Instances of *DatatypeFields* must be created to contain the actual data of a record. The value is held in the Value attribute. (A default value can be set by the user in the Default attribute. This attribute is used during flattening when Value has the value of NULL. The default value of Default is NULL.)

An exception to the DatatypeFields is the Value attribute of the StringPtrField data type. StringPtrField allows the value of a string attribute to reside in the attribute of an object. The Value attribute, called *pValue* in this case, takes an attributepointer to the attribute of an instance that holds the intended string value. Using the StringPtrField datatype means that the data itself does not have to be moved to the Value attribute of the *DatatypeField* instance. The special methods SetPointer() and GetPointer() are provided to manipulate the value of pValue.

After the attributepointer, pValue, has been set, it is possible to use the GetData() and SetData() methods to get and set the attribute's value in its "home" object. These methods use pValue to obtain the original string value.

In general, data in Fields may be accessed directly, regardless of any Record(s) they may belong to, through their accessor, SetData(), GetData() methods.

Note: There is no total length specification; it is calculated just in time.

Record Elements

A record is defined by setting the **Elements** attribute of the Record class to a list of elements (Fields or other Records). The order in which the list is constructed constitutes the structure of the record.

Note: that the Record class is itself an Element. Therefore, this structure allows Record instances to point to other Record instances

Note: Fields and Records may be shared by different (parent) Records. However, including a Record in itself will result in a runtime error during parsing or flattening.

There are two ways to construct a record definition: statically and dynamically. Both of these techniques are illustrated in the Records example in the \Record Handling examples directory.

Static Construction

When employing the technique of static construction, the Aion BRE programmer constructs static instances of the Record and *DatatypeField*s that are necessary to define the desired record structures and data fields at edit time. The Record's Elements attribute is defined at this time as well.

Processing consists of invoking the accessors, *SetValue()* and *GetValue()*, on the statically defined *DatatypeField* instances.

Dynamic Construction

In dynamic construction, instantiating the Record and *DatatypeField* classes is performed by the Aion BRE application at runtime through the standard class *Create()* method. Setting the Record's Elements attribute is accomplished by calling the *SetElements()* method on the record and passing the list of pointers to the record elements as an input argument.

As in the case of static construction, processing consists of invoking the accessors on the *DatatypeField* instances.

Dynamic construction is recommended if two or more records of the same type need to be processed simultaneously, such as in the case of comparing the records.

Serialize Data

To serialize a Record requires the declaration of a buffer of type Binary.

Example:

```
var personBuffer          binary
```

The Record class offers two methods that perform serialization.

Flatten()

Uses WriteString, and WriteInteger, and more, to write the Fields sequentially into a new Binary buffer.

Example:

```
personBuffer = personRecord.Flatten( )
```

Parse()

Uses ReadString, ReadInteger, c., to read a Binary buffer into the attributes mapped by the Fields of the Record.

Example code:

```
personRecord.Parse(personBuffer)
```

Physically writing out the buffer to an external physical medium or populating the buffer with external data must be performed using existing Aion BRE I/O mechanisms, for example, the Read() and Write() method of the File class in IOLib.

MQLib to Access MQSeries

Access to IBM MQSeries is performed using its standard API (MQI). Calls to MQSeries are accomplished using methods in the classes of the MQObject class and its descendents. All invocations first require the instantiation of the owning class. Individual Aion BRE methods result in discrete MQI calls, along with the associated parameters, structures, and common constants for mapping to the MQSeries objects. Structures that are commonly required with a particular MQI call are automatically instantiated, using containment, when the class from which the call occurs is instantiated.

Message queuing allows Aion BRE applications to communicate, using data rather than calls, with any other program that also has access to MQSeries. This includes programs written in Aion BRE, C, C++, COBOL, PL/I, and MVS Assembler, running on appropriate platforms. On certain platforms, MQSeries also provides synchpointing; that is, commit or rollback on logical units of work.

MQSeries support consists of two components: MQLib, a standard Aion BRE library, and *remqnn.dll*, a support .dll file containing all the exported methods of MQLib. Aion BRE does not provide support for MQSeries installation and system administration tasks. These must be provided by your MQSeries administrator. For example, an Aion BRE application can put a message onto a Queue, but it does not create a new Queue.

For information on the classes and methods in MQLib, see the “MQLib” chapter in the CA Aion BRE online help.

Code the Queue Manager

The Aion Queue Manager object, `MQQueueManager`, should be dynamically instantiated as needed, along with each Queue (Aion object `MQQueue`) and Message (`MQMessage`), and can be reused easily. Crucial subsidiary objects are automatically instantiated through the Aion BRE process of containment. For instance, instantiating `MQQueueManager` automatically instantiates the Object Descriptor (`MQOD_ObjectDescriptor`), the Begin Options (`MQBO_BeginOptions`), and the Connect Options (`MQCNO_ConnectOptions`). Instantiating `MQQueue` automatically instantiates the Get Message Options (`MQGMO_GetMessageOptions`), and the Put Message Options (`MQPMO_PutMessageOptions`). Instantiating `MQMessage` automatically instantiates the Message Descriptor (`MQMD_MessageDescriptor`). In most applications, these objects suffice.

When the less common data objects are required, such as `MQTM_TriggerMessage`, users must instantiate them manually.

MQLib Data Objects

MQSeries expects the various data objects to be presented in a C-type structure. Therefore, a conversion is required in both directions between the object construct in which Aion BRE stores, for instance, the Get Message Options, and the structure MQSeries needs. The processes accomplishing these conversions are known within MQLib as *flatten* and *parse*. A hierarchical Aion object is flattened into a sequential C structure for MQSeries' consumption. A returned structure is parsed into the Aion BRE object. The flattened version of the data resides in an attribute of data type Binary, within the data object.

Data objects are provided with a pair of methods. For example, FlattenMQGMO and ParseMQGMO manipulate the Get MessageOptions into and out of the attribute flatMQGMO. Data that only passes in one direction only has the one required method. For those which are automatic, the flatten and parse occur automatically. For example, the MQGet method calls the FlattenMQGMO method before getting the Message, and calls the ParseMQGMO method afterward. Manually instantiated objects such as Trigger Message require user-written invocations of the FlattenMQTM and ParseMQTM methods.

Chapter 8: Process XML

CA Aion BRE provides the native ability to read and generate XML documents that are related to the objects, attributes and values within an Aion BRE application. Where an XML schema is available, an Aion BRE utility using COBSLib can automatically generate the Aion BRE object model that corresponds to the schema, as well as the mechanisms for automatically populating that model based on XML documents that conform to the schema.

Three components of XML are addressed by these features:

- The Simple API for XML (SAX) represents XML data as a sequence of events for simple high-level read-only access. CA Aion BRE provides the SAXLib system library for this support.
- The Document Object Model (DOM) represents XML data as a tree structured document in storage and provides low-level read/write access to individual data objects such as elements and attributes. CA Aion BRE provides the DOMLib system library for this support.
- XML schemas provide a formal description of the constraints and structure of XML documents. This structure can be mapped to the structure of an object model in CA Aion BRE. CA Aion BRE provides the XsdConverter utility for this purpose.

Using these capabilities, you can:

- Access a SAX reader and process callback methods from a SAX reader within an Aion BRE application. The Aion BRE application interacts with the SAX reader by means of SAXLib.
- Use the DOM API from within an Aion BRE application to access, read, update and/or create an XML document: The DOM API is accessed through methods provided by DOMLib.
- The XsdConverter utility can generate an Aion BRE library that corresponds to an XML Schema (.xsd file). XsdConverter also provides automatic facilities for populating instances of classes in that library from XML documents that conform to that schema and for generating XML documents from Aion instances.
- Access the DomExample.App and SaxExample.App example programs. These example programs are available to help you understand how to use Aion BRE to process XML files.. These examples have corresponding help files describing them named DomExample.txt and SaxExample.txt respectively.

This chapter assumes that the reader is familiar with the structure of an XML document and knows what XML elements and attributes are.

This section contains the following topics:

[SAXLib - Read XML Documents](#) (see page 258)

[DOMLib - Read and Write XML Documents](#) (see page 265)

[Generate Applications Based on XML Schemas](#) (see page 273)

[Use the XsdConverter](#) (see page 277)

[Process the XML Document with the Generated Application](#) (see page 278)

[Automatic Unmarshalling and Marshalling](#) (see page 281)

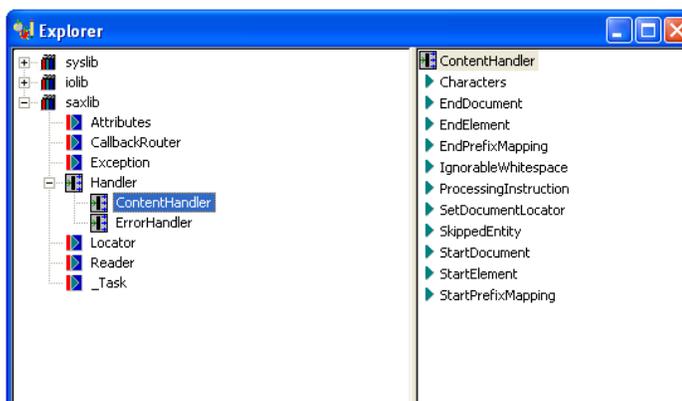
[The Purchase Order Example](#) (see page 282)

SAXLib - Read XML Documents

The SAX API only supports reading XML documents. To write an XML documents requires the use of DOMLib.

The SAX processing model may be less familiar than DOM to programmers because it rests on the use of *callback* methods.

The SAX reader reads the source XML document and sends events to the calling application as it encounters the various aspects of the document. The methods invoked by these events are called callback methods, for reasons that will become obvious as you learn how SAX works. It is responsibility of the receiving Aion BRE application to process or ignore events that it receives from the SAX reader.



The “events” that SAX reader returns to the application are represented by the methods of the Handler interface, which provides an interface for content related events (the ContentHandler) and an interface for error events (the ErrorHandler). (See also corresponding methods in the CallbackRouter class—the relationship between the CallbackRouter and the Handlers in the following section).

Note: that the API calls to the SAX reader are contained as external methods in the `_Task` class in SAXLib. These methods are implemented by underlying C++ functions that call the Apache Xerces SAX API. There is no reason why the Aion BRE user needs to see the methods of the `_Task` class.

How SAXLib Functions

The SAX reader calls the methods of the CallbackRouter. Those methods reroute those calls to your implementation of the corresponding methods in the class that implements the ContentHandler interface through the `CallbackRouter.InvokeHandlerMethod()` class method.

The methods in the following table are provided by the ContentHandler interface from the SAX reader. The order represents the approximate order in which these methods are invoked.

Note: The implementation of these methods must return a boolean value. Returning False indicates that you wish to end the parsing session on the current event by throwing an exception. The exception is thrown from the underlying C++ code that calls the Apache Xerces API.

Method	Explanation: Typical Implementation
StartDocument	This event occurs at the start of parsing the document. It is called only once by the SAX reader. The implementation should perform any initial processing required by your application before the actual parsing of the document begins.
StartPrefixMapping	This event occurs when the SAX parser encounters an element that uses the <code>xmlns</code> attribute to declare a namespace. This process is called <i>prefix mapping</i> . The method is given the prefix and URI associated with the prefix. The typical implementation is to store this information for use during element processing (see the <code>StartElement()</code> method).
StartElement	This event signals the beginning of parsing an element in the XML document. The input to your implementation includes the name of the element (in various forms: URI, local name and QName). Also, when this event occurs, it automatically

Method	Explanation: Typical Implementation
	<p>creates an instance of the Attributes class, a pointer to which will be passed to your method. Once you have the pointer to the Attributes class, you can use the accessor of that class to retrieve information related to the attributes of the element (see Attributes Class).</p> <p>Note: The Attributes instance is created even if there are no attributes for the instance.</p>
Characters	<p>This event occurs when the SAX parser encounters textual data within an element. Your implementation method receives the character string along with a specification of the length of the string. It is the responsibility of your implementation to parse the string.</p> <p>Note: Character data may be reported over several events.</p>
IgnorableWhites pace	<p>This event occurs when ignorable whitespace occurs. What can you do with whitespace? Ignore it! There's not much to do in this implementation, except return True.</p> <p>Note: The same considerations that apply to Characters(), for example, returning the "data" over several events, also applies to IgnorableWhitespace().</p>
ProcessingInstru ction	<p>This event occurs when the SAX parser encounters an XML processing instruction (PI). The target and any data sent in the PI are provided to your app. If the XML document you will be parsing contains PIs, here is where you will provide the specific application code to implement those instructions.</p>
EndElement	<p>This event signals the end of parsing an element. Implementation may involve repositioning your Aion application for the next element in a tree that your application is creating.</p>
EndPrefixMappin g	<p>This event occurs when the SAX parser has concluded processing an element that has required prefix mapping. If the prefix and URI have been stored in the app, this method should remove those settings.</p>
EndDocument	<p>This event occurs at the finish of parsing the document. It is called only this once by the SAX reader. The implementation should perform any final processing required by your application at the end of parsing the document.</p> <p>Note: EndDocument() will also be called after parsing halts due to (fatal) errors in the document.</p>
<p>The following methods are not supported by Xerces. Implementation of these methods should be just: return True.</p>	
SetDocumentLoc ator	<p>When an event of this type occurs, an instance of the Locator class is created and the pointer to that instance is passed as</p>

Method	Explanation: Typical Implementation
	an input to your implementation of the SetDocumentLocator().
SkippedEntity	This event returns to your application any Entity reference that has not been resolve/expanded by the XML parser. The Apache Xerces parser never invokes this method: entity references will always be expanded. So again, your implementation will simply be to return True.

Attributes Class

The Attributes class provides the means of obtaining the names and values of the XML attributes of an element. In your implementation of the StartElement() method you will want to loop through the attributes of the element. For each element, a single instance of the Attributes class. To loop through the attributes of an element you begin by obtaining the number of attributes present for the element: ptrElementAttributes.getLength(). You can then loop through the attributes using a counter that ranges from 0 to the number of attributes as an index to get the name (various versions), type, and value of each attribute. You can also get the index, type and value for a particular attribute if you already know its name.

Note: The Aion application should never call the Create() method of the Attributes, Locator, and Exception classes. These Create() methods are for internal use only by methods of the CallbackRouter.

Process Exceptions

Each of these methods is capable of throwing Exceptions. When an exception is thrown by the SAX reader, an instance of the Exception class will automatically be created and a pointer to it is passed to your implementation of the error methods in your implementation of the ErrorHandler. The ErrorHandler provides three methods that will be invoked via the CallbackRouter:

- Error(): The method is invoked when the SAX parser encounters a recoverable error. In the implementation of the method you may wish to log the error and perhaps exit the parsing process by returning False from the method.

Note: Recoverable errors generally pertain to validation conditions that are violated. SAXLib currently does not provide validation capabilities.

- `FatalError()`: Fatal errors typically relate to a document's not being well-formed and, therefore, necessitate stopping the parsing of that document. The implementation of this method should log and/or display the error and gracefully exit the parsing process by, for example, returning `False` from the method.
- `Warning()`: Warning messages are defined in the XML 1.0 specification and relate to the DTD. Because no provision is made by SAXLib to validate a document in terms of its DTD, there is no need to provide special processing for this method.

Exception processing will probably involve writing or displaying an error message to the user. Use the accessor methods of the Exception class to obtain information regarding the particular exception.

More Information:

[How SAXLib Functions](#) (see page 259)

Use SAXLib

The Aion application must implement two interfaces:

- `ContentHandler`
The implementation of a method of the `ContentHandler` specifies how the application is going to process a particular aspect of the XML document.
- `ErrorHandler`
`ErrorHandler` methods are for processing any error or warning messages that are encountered by the SAX reader.

The following steps are required in constructing an application using the SAX API:

1. Instantiate the application classes that implement the `ContentHandler` and `ErrorHandler`.
2. Instantiate the `Reader` class, which is provided by SAXLib, and "register" the instantiations of the `ContentHandler` and `ErrorHandle` with the `Reader` by passing the pointers to these application instances in the `Reader.Create()` method call.

Note: Creating an instance of the `Reader` also creates an instance of the `CallbackRouter` and registers this instance with the SAX reader.

3. Call the appropriate parsing method of the `Reader` (`ParseFile()` or `ParseString()`), passing to it the appropriate parameter (for example, complete path to the XML document) that the SAX reader is to read.

Use the SAX API

Here is a sample XML code you can use to illustrate the SAX API:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <bookstore>
  <book author="Win Big" isbn="0-00000-000" name="Bigg Honcho" />
  <book author="Hope for the Future" isbn="9-99999-99" name="Hope Moreso" />
  <book author="How to Argue" isbn="6-66666-666" name="Mary Contrarie" />
  . . .
<!-- Other listings of the <book> element -->
</bookstore>
```

Bookstore.xml has a simple structure. The root element is simply `<bookstore>`; it starts with no attributes. A typical attribute might be the name of the book store. The elements of a book store are its books. Each `<book>` element has three attributes, Name, Author, and ISBN number.

The application must contain both IOLib and SAXLib.

The application also is likely to contain the following classes:

- Book class with attributes Author, ISBN, and Name.

The application must contain the following classes.

- An implementation of the ContentHandler
- An implementation of the ErrorHandler.

1. Implement the methods of the Handlers:

Use all the methods specified by the ContentHandler and ErrorHandler on MyContentHandler and MyErrorHandler respectively. All ContentHandler methods must minimally return True and those of ErrorHandler would probably return False. Returning False will automatically terminate the SAX Session in case any of these errors occur.

Code that implements steps 1 through 3 from the *Use SAXLib* section that initializes the process the communication with the SAX reader:

```
var pMyCH is MyContentHandler
var pMyEH is MyErrorHandler
    // For automatic deletion, use "contained instances"
    // for these variables.
pMyReader is &Reader
pMyReader = Reader.Create
pMyReader.Initialize(null, pMyCH, pMyEH)
var filepath is &DirectoryEntry
filepath = FindFile(FILE_NAME)
if (filepath <> null) then
    pMyReader.ParseFile(FILE_NAME)
else
    // Signal a file-not-found condition.
End
pMyReader.delete()
```

This code goes directly from calling the SAX reader to deleting the reader. What happens between reading the file and deleting the reader? The application should be reading the XML document and creating instances of the Book class. How do the instances of Book get created? The answer lies in how the callback methods work. Procedurally, the program is processing the parsing method (synchronously), while the callback methods are being called as the SAX reader encounters different aspects of the document it is being read.

The creation of the Book instances must be implemented by the StartElement() method. This method needs to (1) construct and instance of the Book class, and (2) populate the attributes of that instance from the XML attributes of the element being read.

2. A typical implementation to StartElement(). Remember that StartElement() is called each time the reader encounters a new element.

Note: The first element to be read by the parser will be the <bookstore> element, in which we are not interested.

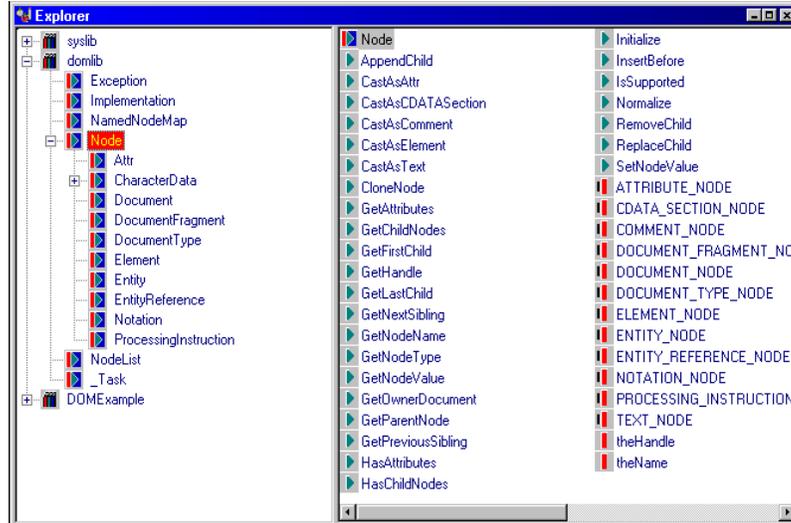
```
var pBook &Book
var len is integer
// We can ignore the root element QName = "bookstore"
if QName = "book" then
  pBook = Book.Create()
  pBook.setName(attrs.GetValuebyQualifiedName("name"))
  pBook.setAuthor(attrs.GetValuebyQualifiedName("author"))
  pBook.setIsbn(attrs.GetValuebyQualifiedName("isbn"))
end
return True
```

where attrs is the pointer to Attributes instance for this element that is passed to your implementation.

DOMLib - Read and Write XML Documents

The DOM model allows the entire XML document to exist in memory at one time as a tree structure. The central concept is that of a Node: every distinct aspect of an XML document, for example, its elements and attributes, is represented as a Node in this tree. Thus, when you open DOMLib in Aion, you will see that the Node class is the parent class of a hierarchy that includes Document, Element, Attribute (the Attr class), Entity, Text, and other aspects of an XML document. Each of these aspects is considered to a Node in a DOM tree that represents the document.

The class structure of DOMLib precisely corresponds to the DOM metamodel of XML documents presented in the standard XML literature.*



Note: Attr is itself a Node, just as Element is a Node. Thus, the structure of a DOM tree differs from the structure of an XML document. However, as you see, this feature is not explicitly revealed by the DOM API, which preserves the “appearance” of an XML document.

The Node class offers various operations to query a node (for example determining how many child nodes it has), to get the “data” from a node (for example., retrieving the name of the node), and to manipulate a node (for example., removing a child node). These methods are inherited by each type of particular node, for example., Document and Element. The Node class also provides the constants for identifying the type of Node currently being read.

Note: As with SAXLib, the external functions calls to the DOM API reside in the _Task class. There is no need to look at the _Task class.

In addition to the Node class, the DOM model consists of four other classes:

- Exception: Provides the code of an exception. Also provides the constants for DOM errors.
- Implementation: Every DOM document must have an Implementation. Think of the Implementation as establishing a DOM “reader”.
- NamedNodeMap: Provides handle to the attributes of a Node (element).
Note: This class functions similarly to the Attributes class in SAXLib. It is created from the Node.GetAttributes() method, which retrieves the Attr nodes of an element, and is traversed in the same manner as the Attributes class.
- NodeList: Provides a handle to the list of all nodes in the XML document.

The Aion BRE methods corresponding to the DOM API are distributed over the classes that constitute the DOM model.

Initialization

To read or create an XML document (under DOM), you need to obtain a pointer to the document. The document as a whole is called the Document node. Two pointers must be defined:

- ptrImplementation is &Implementation
- ptrDocument is &Document

Note: In addition, to retrieve information from this document it is necessary to obtain the root node (element) of the document. To do this you must define a third pointer:

- ptrDocumentElement is &Element

Initialization code is typically the following:

```
ptrImplementation = Implementation.Create()
var filepath is &DirectoryEntry
filepath = FindFile(FILE_NAME)
if (filepath<> null) then
    ptrDocument = ptrImplementation.ReadDocumentFromFile(FILE_NAME)
else
    ptrDocument = ptrImplementation.CreateDocument(null,"bookstore", 0)
end
ptrDocumentElement = ptrDocument.GetDocumentElement()
```

PtrDocumentElement points to the root node of the document, which in this case is the <bookstore> element.

The Else-condition creates a document with the root node of <bookstore>. This allows execution to proceed without error if there is no document-retrievals of books from this created document will just yield an empty list. Document creation will be discussed in the section *Create an XML Document*.

Process the DOM Tree

A typical strategy for using DOMLib is to loop through a document tree, testing each node as to type and processing the Node appropriately. The major difference between the SAXLib implementation and the DOMLib implementation is that, in the latter, processing of elements must occur with explicit looping within the method. The code examples here are extensively commented for explanatory purposes (you may want to omit the comments from your code).

The following code is DOM code that performs the functions of the code discussed in the section *Use the SAX API* in this chapter.

```
var listBooks is list of &Book
    // From the original SAXLib implementation of this method.
var pBook is &Book          // Originally used in StartElement().
var length, count is integer=0
    // The following objects are created corresponding to objects
    // retrieved by the DOM reader from the XML document.
var MyNode is &Node
var MyBook is &Element
var myNodeList is &NodeList
myNodeList = ptrDocumentElement.GetChildNodes()
    // Create the NodeList instance for looping through the document.
    // Note: a NodeList is just a Handle into the DOM model.
length = myNodeList.getLength()      // Retrieve the number of nodes.
Loop
    breakif count = length
    myNode = myNodeList.Item(count)
```

```

        // The Item method of NodeList allows retrieving each
        // node. (See the Index() method of Attributes.
if (myNode.GetNodeType() = ELEMENT_NODE) then
    // Test the type of node retrieved. Ignore it if it
    // is not an Element node.
myBook = myNode.CastAsElement()
    // It is necessary to cast the retrieved node as an
    // Element so that the methods provided by the
    // Element class can be used. Given the preceding
    // node type test, this recasting is safe.
pBook = Book.Create()
pBook.setName(myBook.getAttribute("name"))
    // Set the value of the Name attribute of the Aion
    // object by retrieving (the value) of the name
    // attribute of the XML element.
pBook.setAuthor(myBook.getAttribute("author"))
pBook.setIsbn(myBook.getAttribute("isbn"))
end
count = count + 1
end

```

Note: The method used to retrieve the nodes is `GetChildNodes()`. Remember `ptrDocumentElement` points to the root element, `<bookstore>`. Therefore, the first node that is retrieved by `GetChildNodes()` will be, in fact, the first `<book>` element. However, `GetChildNodes()` does *not* return the attributes of a node, in this case, attributes of the `<bookstore>` node, as child nodes, even though these are technically children of the node just as are its subelements!! In this way the DOM API preserves more of the structure of the XML document than it reveals of the structure of the DOM tree itself. However, `GetChildNodes()` *does* return Character Data nodes (comments, ordinary textual insertions, and CDATA sections, as child nodes of the parent element node (along with any subelements, of course).

XML Maintenance Using DOMLib

Unlike SAX, DOM also provides APIs for updating and even generating an XML document. You can create methods to add and delete elements.

Add Elements to an XML Document

Following the previous example of `bookstore.xml`, you can use DOMLib to add additional `<book>` elements to the document. Assuming input to the process as follows:

```

in author is string
in isbn is string
in name is string

```

Code to add a <book> element is as follows:

```
var myElement &Element
myElement = ptrDocument.CreateElement("book")
myElement.SetAttribute("name",name)
    // Where the quoted occurrence of name refers to the (name of
    // the) attribute in the XML Element and the name variable refers
    // the value of Aion object attribute called 'name'.
myElement.SetAttribute("author",author)
myElement.SetAttribute("isbn",isbn)
```

Include the created Element in the document by using Node's `appendChild()` method to append it to the appropriate node. In this case, the <book> elements all "hang off" the root node, <bookstore>, of the document, so you need to append this new element directly to the document's root node:

```
ptrDocumentElement.appendChild(myElement)
```

Finally, write completed XML document back to the bookstore.xml document by using the `Document.WriteDocumentToFile()` method.

```
ptrDocument.WriteDocumentToFile(FILE_NAME)
```

Delete Elements from an XML Document

To delete a <book> element from bookstore.xml, you will need to accept one of the attributes of the <book> so that the appropriate element node can be identified.

Example:

```
in name is string
```

The implementation of this method is very similar to the implementation in preceding Processing the DOM Tree. You need to loop through all the nodes of the document looking for one particular node-the one with the attribute matching the input parameter. At that point you can use the `Document.RemoveChild()` method to delete that element. The code can be:

```
var myNodeList is &NodeList
var length, count is integer=0
var MyNode is &Node
var MyBook is &Element
myNodeList = ptrDocumentElement.GetChildNodes()
length = myNodeList.getLength()
Loop
    breakif count = length
        myNode = myNodeList.Item(count)
        if (myNode.GetNodeType() = ELEMENT_NODE) then
            myBook = myNode.CastAsElement()
            if (myBook.getAttribute("name") = name) then
                ptrDocumentElement.removeChild(myBook)
                break
            end
        end
    end
    count = count + 1
end
```

Create an XML Document

You might want to create an XML document in order to marshal instances in your main Aion application. For example, writing the instances and their attribute values to an XML document. Initializing a document to be created follows the same preceding steps, with the exception, of not reading the document from an external file.

1. Create an instance of Implementation.
2. To create a document use the `CreateDocument()` method of Implementation.
 - a. Pass namespace URI as the first argument.
 - b. Pass the desired name of root node as the second argument
 - c. Pass "0" (zero) or NULL as the third argument (document type).
3. Get the root element of the document, which is again accomplished by calling the `GetDocumentElement()` method.

The preceding Initialize section as coded satisfies these steps if the FILE_NAME is not found in the current directory.

Marshalling becomes a matter of looping through the instances in the Aion application and generating the appropriate elements. A simple strategy would be to use the Element generation code used in the section Adding Elements to an XML Document and accessing the value of the attribute from a given instance of Book.

Example:

```
var myElement &Element
var pBook &Book
For Book, pBook
myElement = ptrDocument.CreateElement("book")
myElement.setAttribute("name", pBook.GetName())
...
ptrDocumentElement.appendChild(myElement)
end
```

Note: It is a good design principle for Aion classes to “know” how to marshal themselves. Each class should contain its own marshalling method. In the preceding code example, the loop might consist of a single line of code: pBook.Marshal(...). (The method will need to accept a pointer to its parent element as a parameter.)

Subelements can be created from Aion instances that are referenced (pointed to) by the instance that represents the parent element. Create an element for the referenced instance and append to the parent element.

Note: An alternative means provided by the DOM API for creating attributes for an element is to use the Document.CreateAttribute() method to create an attribute that is unattached to any element and then to use the Element.SetAttributeNode() method to associate the attribute with an element.

Handle Character Data as Element Values

There are many styles in which XML documents can be written. In particular, another style involves

- Treating the attributes of the book element `<book>` (Author, ISBN, Name) as subelements of `<book>`.
- Including character data within the element, for example, `<author>John Q. Author</author>`.

Handling subelements is just a matter of looping within a loop and testing for the name of the element. Retrieving the value of an element is accomplished under DOM by means of the `getNodeValue()` method. SAX will throw a `Characters()` event upon encountering the string of characters.

Generate Applications Based on XML Schemas

An XML Schema is used to define valid structure of an XML document. It declares what structure and content are allowed in the document. Some XML parsers can confirm that a document obeys the constraints of the Schema.

What makes a Schema useful in Aion BRE is the subset of it that defines the structure only, not the content. Aion BRE provides a utility, `XsdConverter`, that reads the Schema and uses `COBSLib` to generate an application file representing the XML data in Aion classes and attributes. Also generated within these classes are methods for loading the data from a DOM into Aion objects, and dumping the Aion objects back out to the DOM. Control methods are generated to read an XML file into the DOM, and write it back out. And finally, accessor methods are provided for all attributes. This generated application can then be included, as a library, in a user-written application, to provide an interface to an XML document that complies with the Schema.

General Approach

The schema itself is an XML document, and consequently can be read using SAXLib. SAXLib is preferred over DOMLib, because it sequentially reads the entire DOM, so nothing gets overlooked. Using DOMLib would require considerable additional logic to control the flow of access, which is automatic in SAXLib. The cost is a little more control logic: one stack to keep track of positioning in the Schema being read, and a second stack to keep track of positioning in the Aion application being written. Thus each object in the Schema is sequentially evaluated, and appropriate Aion classes and attributes are created.

After the Schema has been completely processed, XsdConverter then evaluates the generated classes and attributes, in order to create four types of methods:

1. CONTROL methods. These are always the same, regardless of what has been generated from the Schema. An internal class called `_Control` is created, with some DOM control attributes, and methods to Create the Control class, Read From File Into DOM, and Write from DOM Out to File.

The specialized `Create()` method also instantiates the DOM Implementation class.

To read an XML document into memory, execute the `Read()` method, passing it the name of the .xml file you wish to read. This method establishes the pointer to the Document (which is available through the `GetDocument()` accessor method) and returns the pointer to the document's root element node (see `ptrDocumentElement` in the preceding example.)

To save an XML document, execute the `Write()` method.

The `_Control` class also provides a specialized `Delete()` method that removes the instances of Implementation, Document, and the root element node that it has created for you behind the scenes.

2. LOAD & DUMP methods. Each class is given a method named `Load`, which accesses the DOM, this time via DOMLib, obtains the appropriate data, and populates the attributes of the class. For attributes that are pointers to other classes, the `Load` method of the other class is invoked. Thus the user may call `Load` on the topmost class, and the entire structure will get loaded. Inverse methods, named `Dump`, are created to move the data from Aion to the DOM.
3. CREATE & DELETE methods. A similar cascade of `Create` methods are written, which allows the user to call the `Create` of the topmost class, and all the following attribute pointers are traversed to instantiate all the classes comprising the data structure. The `Delete` methods make sure that subordinate classes get deleted when their superior class is deleted. Both `Create` and `Delete` are specializations of the native classes so named.
4. ACCESSOR methods. A complete set of getter and setter methods are also produced.

More Information:

[Automatic Unmarshalling and Marshalling](#) (see page 281)

Details

1. One instance of `_Control` is needed for each behind-the-scenes DOM, so typically one instance will be needed per file.
2. All access to the XML document being read is through `DOMLib`.
3. A generated application will contain a class for each element that contains other elements or has a datatype. The class will have attributes representing each attribute on the element, as well as attributes containing pointers to other classes for contained elements. Contained classes which are flat become attributes of the containing class.
4. In an XML document both element attributes and also elements themselves (when empty and/or with a datatype) become aion attributes. To know where to get or put this data, attributes must be marked to indicate their source; this is done in the attribute comment, with "[XML Attribute]" and "[XML Element]".

Note: that this flag is only used at generation time, resulting in different Load and Dump logic. All Aion objects have a standard comment (appended if the previous comment is in use), specifying the source Schema and a timestamp.

5. In general, schema objects (or portions of objects) which define data type or structure are mapped, while those which validate are ignored. It is presumed the schema is in fact in place and being used, and will provide the validation.
6. Schema elements may directly contain data, whereas Aion classes may not, requiring an attribute to hold any value. When mapping such elements, an Aion attribute named `_data` is added to the Aion class, to hold the value.
7. Namespaces are not supported at this time.
8. As far as possible, Aion objects are given the same name as the Schema objects they represent. When the Schema object has no name, one is generated from its type. Schemas allow duplicate names in places where it is disallowed in Aion. Typically this occurs when a reference to an object is given the same name as the object itself. Such references become pointers to objects in Aion, the duplicate name is detected, and the name of the pointer is made different by prefixing it with a "p". Undetected cases (there are none currently known) will generally result in an Aion object being created with a trailing underscore added to the name, and the code will still work. There remains the possibility that invalid duplicate names will be attempted such that the object is not created. This will result in generated code that has missing pieces and does not build correctly.

9. Here is the mapping between Schema and Aion BRE objects. All Schema objects missing from this list are ignored.

Schema	Aion
Attribute	Attribute
AttributeGroup	"Virtual" Class1
ComplexType	Class
Element	Attribute or Class2
Enumeration	Constant Attribute
Fixed=	Class-level=TRUE
MaxOccurs=	If > 1 or unbounded, Attribute type is List Of specified type3
Name=	Object name
Ref=	Attribute of type Pointer to class
Restriction.base=	Attribute type
Sequence	[ignored]4
SimpleType	[ignored]4
Type=	Attribute type1

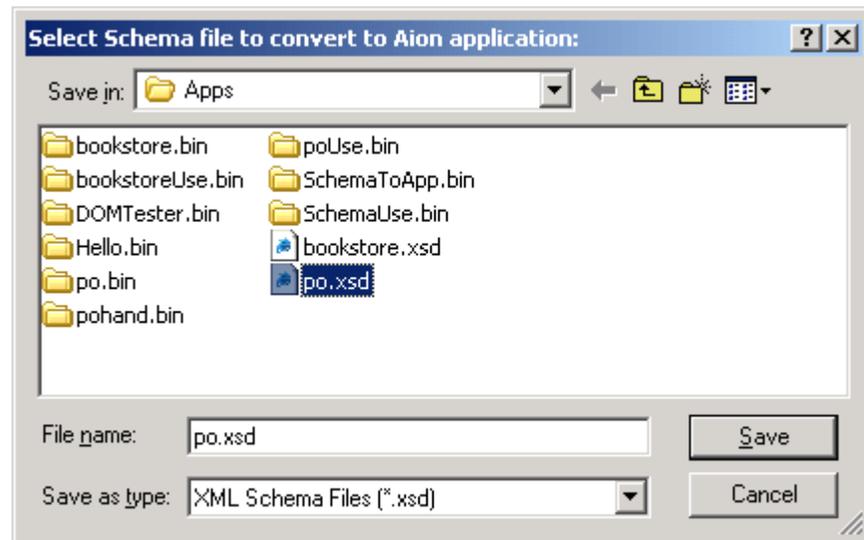
Additional Notes

1. An AttributeGroup generates a temporary Aion class containing its attributes. After the entire Schema has been processed, the generated application is scanned for attributes that are pointers to such classes, which are replaced by the attributes in the class. Then the temporary class is deleted.
2. Schema elements with contained subelements become Aion classes. Elements without subelements become attributes. If the Schema element has an attribute named type set to a value of a datatype, this datatype is translated into an Aion datatype for the Aion attribute. Otherwise the type must indicate another class. In this case, the one Schema element results in two Aion objects: a new class named after the type, and an attribute in the superclass, pointing to the new class.

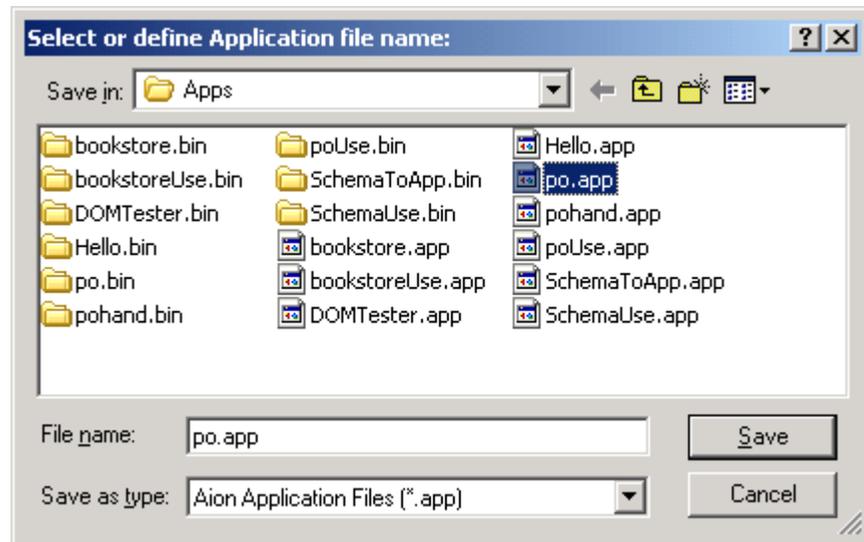
3. Lists represent a variable number of instances of the subordinate class. At Create time the number is unknown, so no classes are created. If new data is being created, the user application must instantiate such classes manually. At Load and Dump time, instances (Aion or Schema) are automatically created dynamically and populated by looping through the source data. Delete also loops similarly.
4. Sequences and SimpleTypes necessarily contain further data, which together with any container data, are sufficient for building the Aion data structure, so these two Schema objects are ignored.

Use the XsdConverter

A Schema file, typically of filetype XSD, is the single input required to run `_xsdConverter.exe`. The single output is an Aion BRE application file. To run `_xsdConverter.exe` specify each name in the Open File dialogs presented:



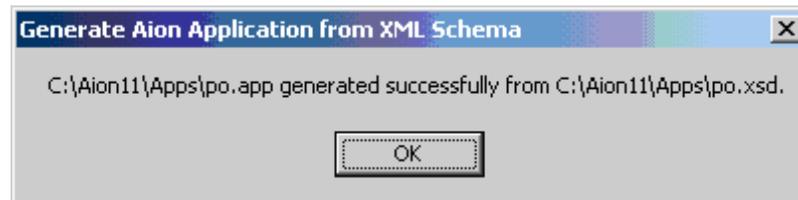
Note: the example documented here uses the po.xsd file. The example files that are provided now are xsdTest.xsd and xsdTestLib.app.



Important! When creating new files, the file extension (.xsd or .app respectively) must be specified.

After these two specifications are made, the application will run for a while, and announce success with a message box like this:

Figure 1: Show Generating aion Application from XML Schema. Click OK.



The generated application file, in this example po.app, is then included, as a library, in the user-written application that will be accessing the XML document. The generated application should be manually restored from source first, then the user application should be restored from source.

Process the XML Document with the Generated Application

A generated application can be edited in the Aion BRE IDE as necessary. Then the application can be used to process XML documents.

Read an XML Document

1. Instantiate the `_Control` class.
2. Invoke the `Read` method of this class, specifying the XML file to read. This creates an XML DOM and populates it from the XML file, returning the Document Element of the DOM.
3. Instantiate the `Aion` class which corresponds to the topmost class in the Schema, usually corresponding to the `Document`. All subordinate classes will be automatically instantiated as well.
4. Invoke the `Load` method of this class, specifying the Document Element returned by `Read`. All subordinate classes will be automatically instantiated as well. Access the data in any loaded class via its `Get` and `Set` methods.

Example:

```
myControl = _Control.Create()

myPORoot = myControl.Read("C:\Aion10\Apps\po.xml")
myPO = PurchaseOrder.Create()
myPO.Load(myPORoot)
myItems = myPO.GetpItems()
myItemList = myItems.Getpitem()
MessageBox("Item #2=" & myItemList(2).GetProductName())
```

More Information:

[Automatic Unmarshalling and Marshalling](#) (see page 281)

Write an XML Document

1. Instantiate the `Aion` class which corresponds to the topmost class in the Schema.
2. Populate the `Aion` class and its subordinates with the `Set` methods.
3. Instantiate the `_Control` class. One instance can be used for both reading and writing.
4. Invoke the `Dump` method of the topmost class, specifying the desired Document name, and the `_Control` instance. A Document is created in the DOM, and fully populated from the topmost class and its subordinates.
5. Invoke the `Write` method of this class, specifying an XML file to receive the data.

Example:

```
myControl = _Control.Create()
myPO = PurchaseOrder.Create()
myPO.SetorderDate(CurrentTime())
myPO.GetbillTo().SetCity("San Francisco")
myPO.GetbillTo().SetCountry("USA")
myPO.GetbillTo().Setname("Joan Smith")
myPO.GetbillTo().GetState().Set_data(CONSTANT_22)
myPO.GetbillTo().Setstreet("7 Elm Street")
myPO.GetbillTo().Setzip(94133)
myPO.Dump("PurchaseOrder", NULL, NULL, myControl)
myControl.Write("C:\Aion10\Apps\po.xml")
```

Update an XML Document

1. Follow the sequence for reading. See the "Read an XML Document" section.
2. Update data in Aion structure with Set methods.
3. Continue with sequence for writing at Step 4.

Note: that output may be written to the same or a different file.

Notes:

- Output can be written with a different Document name than specified in the Schema. Such data cannot be read with the generated Aion BRE application, which requires the original name.
- The generated application can be edited in the Aion IDE as necessary.
- A subset of the Aion BRE data structure may be created, deleted, loaded, or dumped, by invoking that method on a subordinate class.

Automatic Unmarshalling and Marshalling

When you generated an application from an XML Schema through the XsdConverter, mechanisms are added to the generated classes that simplify the handling of the relationship between the XML documentation and Aion classes. These mechanisms address:

- Automatically populate instances in the Aion BRE application from XML documents corresponding to the XML Schema.
- Marshal the objects of the Aion BRE application into an XML document using facilities provided by the generated library.

Note: A subset of the Aion data structure may be created, deleted, loaded, or dumped, by invoking that method on a subordinate class.

The domain classes generated by XSDLib reflect the elements defined in the XML schema where the XML attributes are attributes of the generated Aion class and subelements are accessed via attributes of the type "pointer to...". In addition to the standard Set/Get accessor methods, XSD also provides each domain class with methods to load and marshal the class.

Load() Method

The Load() method unmarshalls an element and all the subelements of that element. You pass the method the pointer to the Element (of DOMLib) that was created as the result of parsing the XML document.

Example:

```
mybookstore.Load(myBSRoot)
```

Besides also creating all the Book instances of Bookstore, the Load() method automatically unmarshalls the values of attributes from the element's attributes (using the NamedNodeMap for the element) as well as create and populate the classes corresponding to any subelements of the element. Since the <book> element doesn't contain any subelements, loading books from bookstore.xml, will be a simple process. But where the main element is complex, having the Load() method can save significant programming effort.

Dump() Method

The Dump() method marshals an instance of class, along with any instances of objected that are referenced by the attributes of that instance. Invoke it on each instance that you wish to hang an element off the root node of the document. The Dump() method takes four parameters: the name of the document, the pointer to the document, the pointer to the element to which the generated elements to which the generated elements are to be appended, and a pointer to the _Control instance.

Note: When call the Dump() method, it is necessary to pass only the name of the document and the _Control instance pointer:

```
pBook.Dump("bookstore", NULL, NULL, p_Control)
```

If the Document pointer is Null, this method will create a Document whose root element is the name passed to the method.

The Document and Element parameters (the second and third parameters) are used by the generated code when Dump() is called on referenced instances.

The Purchase Order Example

An example has been created using the Schema presented in the www3 Schema primer, a purchase order. The www3 XSD has been copied exactly as po.xsd. Run XsdConverter specifying purchaseOrder.xsd as input and po.xml as output. The user-written application is poUse.app, which includes po.app. Running it first creates some data, which is written out to po.xml, and then immediately read back in and confirmed. Messageboxes confirm success; po.xml may be reviewed with a browser for additional assurance.

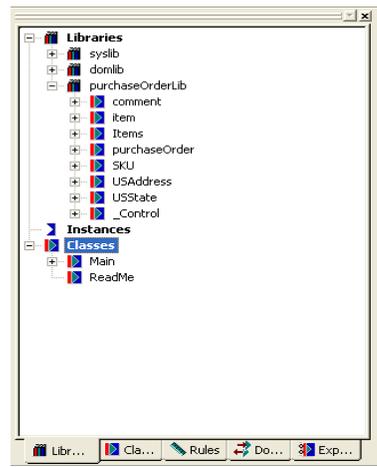
The Purchase Order example provides an advanced schema with several interesting features. Purchaseorder.xsd is the reference schema adopted by the industry. It is expected that the Aion BRE user will run XSDLib on this schema. The xsdTestRun application file is provided to run the generated application and purchaseOrder.xml provides some idea data that conforms to the schema.

Open and study purchaseorder.xsd. Observe that it makes use of both complex and simple type definitions along with an attributeGroup. In addition, purchaseorder.xsd specifies two simple types as enumerations: the shipby and USState types.

Now, following the steps outlined in the Generate Applications Based on XML Schemas section, generate a library (PurchaseOrderLib) with the XSDLib. Be sure to open the generated library first in the Aion BRE IDE, and restore it if necessary to remove error messages. Now open the xsdtestrun application in the Aion BRE IDE and include the PurchaseOrderLib that you just generated.

Compare the class structure of generated library with the structures defined in purchaseorder.xsd.

Notice that each element and type definition of the schema is generated as a separate Aion BRE class, except for the XSD globally named type PurchaseOrderType. That is because the PurchaseOrderType does not play any role in the XML documents that are compliant with this schema. Globally defined named type definitions are equivalent to locally defined anonymous type definitions. XSDLib treats globally defined type definitions as locally defined anonymous definitions.

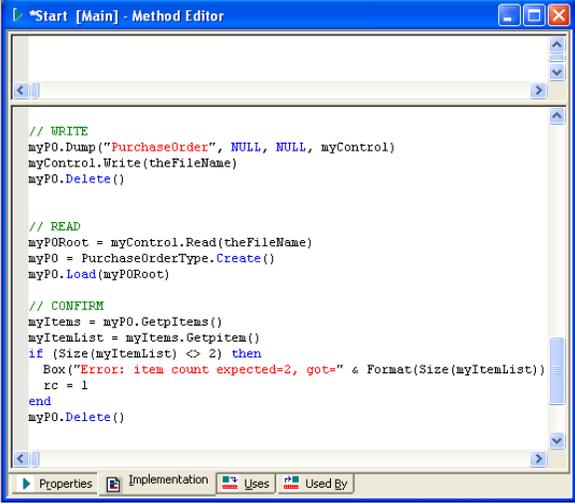


Expand the classes in PurchaseOrderLib and study how the XsdConverter generated the Aion BRE application from the schema. Note that except for the preceding case mentioned, schema elements and attributes that have a complex type are mapped to Aion BRE attributes of type pointer/list of pointer to.

AttributeGroups are assimilated into the element from which they are referenced (see the attributeGroup ref in the item subelement (of Items). The generated Dump() method for the owning classes remembers the AttributeGroup structures and generates the appropriate AttributeGroups from the Aion BRE class.

Enumerated types are mapped to constants in the respective class.

XSDLib generates an attribute, _data, for those types that define no other sources of data (subelements or attributes), see for example comment, SKU, and USState.



```
Start [Main] - Method Editor

// WRITE
myPO.Dump("PurchaseOrder", NULL, NULL, myControl)
myControl.Write(theFileName)
myPO.Delete()

// READ
myPORoot = myControl.Read(theFileName)
myPO = PurchaseOrderType.Create()
myPO.Load(myPORoot)

// CONFIRM
myItems = myPO.GetpItems()
myItemList = myItems.Getpitem()
if (Size(myItemList) <> 2) then
  Box("Error: item count expected=2, got=" & Format(Size(myItemList))
  rc = 1
end
myPO.Delete()
```

Import the generated .app file into xsdtestrun.app. Run xsdtestrun using the purchaseorder.xml file.

More Information:

[Generate Applications Based on XML Schemas](#) (see page 273)

Chapter 9: Domain Interfaces and Dynamic Rules

In addition to Web-based rule management using Aion Rule Manager (described the previous chapter), CA Aion BRE also supports business rule maintenance by means of *dynamic rules*. Dynamic rules allow the business expert to maintain Aion rules outside an Aion development environment by using the Dynamic Rule Manager. Dynamic rules enable CA Aion BRE to process rules at runtime that are not a compiled part of the knowledge-base executable. For more information on maintaining dynamic rules and the Dynamic Rule Manager, see Dynamic Rule Manager in the “Maintaining the Dynamic Rulebase” chapter in the *CA Aion BRE Rules Guide*. Currently, *decision tables* are the only supported form of dynamic rules.

This chapter describes Aion BRE dynamic rules, some of their uses, and the roles and responsibilities of the two human roles in the Dynamic Rule scenario—a *domain expert* and an Aion BRE application developer. When rules are stored in an external medium, such as a database, there must be a way to link the objects in the external medium with the structures in Aion BRE. This link is provided by the *domain interface*. Therefore, we begin this chapter with a discussion of domain interfaces and then discuss how domain interfaces support dynamic rules.

Note: Domain interfaces have a broader use in CA Aion BRE than simply supporting dynamic rules. Domain interfaces can also be used in constructing static (compiled) decision tables. For more information on decision tables, see the chapter “Decision Tables” in the *CA Aion BRE Rules Guide*.

This section contains the following topics:

[Domain Interfaces](#) (see page 286)

[Dynamic Rules](#) (see page 291)

[External Rules: Use Dynamic Rules or Generate Static Rules with COBSLib?](#) (see page 295)

[Styles of Rules](#) (see page 297)

Domain Interfaces

What is the domain interface? In general, interface is a means of exposing functionality while hiding implementation. Besides generating interfaces such as COM, CA Aion BRE provides another way to expose functionality that can be used by domain experts to construct their business rules. COM interfaces are used by components within the system environment; domain interfaces are used by humans. Every Aion BRE application (whether a standalone application or a library) can have (at most) one domain interface.

What does a domain interface consist of? To answer this question we must first consider what a *domain* is. Intuitively, a domain is an area such as an area of business or a type of problem that is the province of an expert. Some examples of traditional expert system domains include automobile diagnosis and the diagnosis of a particular disease. In the business environment, a domain might be a functional area such as loan approval or human resources (for example, matching employee qualifications to open positions) about which rules can be written. Domains may be defined as broadly or narrowly as is appropriate to the task at hand.

A domain requires a vocabulary which is used to talk about the domain. The principal goal of having a vocabulary is to be able to write rules that govern the domain. Thus, we can define "domain" as follows:

Domain

The domain is the name given to a vocabulary and set of rules that describe an area of human concern, for example, a critical business process or decision making task. The vocabulary supports the expression of rules about this domain.

The domain interface of an Aion BRE application consists of the vocabulary that the Aion BRE application provides for the expert to talk about the problem domain addressed by the knowledge base. The domain interface is usually designed jointly by the domain expert and application developer.

However, the domain expert may know little or nothing about Aion BRE or object-oriented programming. (Here we see the principle of interfaces hiding implementation.) The Aion BRE application developer may know little or nothing about the domain expert's area of expertise. Although these two players may work almost independently of one another, they need to have an established common vocabulary between them. This vocabulary serves as a *contract*. Thus, we can provide a general definition of an Aion domain interface as follows:

Domain Interface

A unique type of interface provided by Aion BRE applications that defines the contract between the Aion BRE application developer and the domain expert that allows the domain expert to formulate rules outside of Aion and still have Aion understand these rules at runtime. Every Aion BRE application can have one domain interface.

What does the domain interface look like from the perspective of the application developer? We have said that the domain interface consists of the domain vocabulary that links the way a domain expert talks about a domain with structures in an Aion knowledge base. An Aion knowledge base consists of classes and their methods and attributes. In an Aion BRE application, the vocabulary links to methods.

A method that supports the vocabulary of a domain interface is called a *domain interface member*.

Domain Interface Member

A method that can be referenced through a domain interface. A method designated as a domain interface member represents part of the “semantics” of the domain interface.

Domain interface members can be class methods, instance methods, or `_Interface` methods. They are often Get or Set accessor methods. Such methods are conveniently used in rules to get attribute values for conditions or to set attribute value in actions.

The Method Editor provides a means to designate a method as a domain interface member.

Note: Not all methods may be as designated a domain interface member.

When designating a method as a domain interface member, the application developer gives the member a name called a *domain interface label*. This name is the link to this method from the domain interface.

Domain Interface Label

The name given to a domain interface member for purposes of forming the domain interface (the vocabulary).

Domain interface labels and the functionality they represent must be carefully chosen by the domain expert and the application developer to form the contract for constructing Aion dynamic rules. Domain interface labels should be defined in the domain expert's (business) language. These *labels* are used to by the domain expert to construct dynamic rules (see Dynamic Rule Manager in the "Maintaining the Dynamic Rulebase" chapter of the CA Aion BRE *Rules Guide*).

The collection of domain interface labels given to internal methods constitutes the content of the domain interface. Thus, we can give a more specific definition of a domain interface than the one given above as follows:

Domain Interface

The collection of domain interface labels (business vocabulary) that the Aion BRE application makes available to the domain expert.

Both the general definition and this specific definition are equally valid. The domain interface specifies application resources available to the domain expert; these are the only application resources available to the domain expert.

More Information:

[Dynamic Rules Task Flow](#) (see page 293)

[Create Domain Interface Members](#) (see page 290)

[Domain Interface Member Restrictions](#) (see page 291)

Role of the Domain Interface in System Development

The preceding discussion highlighted the need for the domain expert who is going to create and maintain dynamic rules and the Aion BRE application developer to come to a mutual understanding (contract) regarding the vocabulary and its semantics that will be used to construct the rules of the domain. This section briefly describes how the process of creating the contract (specifying the domain interface) can be integrated into the typical Aion system development process.

Requirements Analysis

Every project requires a Requirement Analysis phase in which the relationship between the system user and the system is precisely defined. Using dynamic rules adds a new factor to this phase. In particular, using dynamic rules requires defining the domain interface. The domain interface must be represented as part of the general system interface that it is defined during this phase.

Elements specified in the domain interface are the domain interface labels that will be assigned to methods in your Aion BRE application. Therefore, these elements must be included in the system dictionary that should be developed during Requirements Analysis and where the semantics of element (what the method in the Aion BRE application will do) would be indicated.

As part of defining the domain interface, the domain itself must be defined and *delimited*. A domain may be defined broadly or it may be defined narrowly and include only a subpart of the total vocabulary necessary to describe the total problem or business area. Aion supports decomposing a problem or business area into separate, specific domains. Although there can be only one domain interface per application, the domain interface may be organized into separate and specific domains on the domain expert's side in the external medium in which the dynamic rules are stored. A single knowledge base can access several domains, and domains may be shared among knowledge bases.

More Information:

[Dynamic Rule Management](#) (see page 303)

System Design

A system design phase usually consists of specifying the classes that will belong to the Aion BRE application.

During system design, the elements the domain interface (the domain interface labels) should be mapped to methods of your classes.

Note: The elements may be mapped to attributes in your *specifications* as long as it is understood that accessor methods must be defined for these attributes in the implementation.

System Testing

In order to conduct sufficient system testing prior to system implementation, it will be necessary to transfer the domain interface labels that have been associated with methods in the Aion knowledge base to an external medium so that the domain expert can construct rules outside the Aion BRE IDE. Dynamic rules need to be tested just as do rules written with the Aion knowledge base itself.

CA Aion BRE provides a tool that transfers domain interface labels defined in an Aion BRE application to a physical database: the Dynamic Rulebase Administrator. The Dynamic Rulebase Administrator should probably be used as soon as the classes and methods required by the domain interface are coded in the Aion BRE application.

Maintaining the Domain Interface

Obviously, as the Aion BRE knowledge base is developed and after it is deployed, the requirements for the application's domain interface may change. Maintaining the specification for the domain interface requires communication between the domain expert and the Aion BRE application developer. As the domain expert discovers new things that "must be talked about" in order to construct appropriate rules for the domain, the application developer must provide the proper semantics within the Aion BRE knowledge base.

Maintaining the domain interface also requires that the external rulebase be kept in sync with the Aion BRE application. As new domain interface labels are introduced into the Aion knowledge base or old ones deleted or modified, this information must be transferred to the external medium. The Dynamic Rulebase Administrator also provides convenient mechanisms to keep the rulebase and Aion BRE application synchronized, see Dynamic Rulebase Scenarios in the "Maintaining the Dynamic Rulebase" chapter of the *CA Aion BRE Rules Guide*.

Create Domain Interface Members

The Aion BRE application developer designates selected methods as belonging to the domain interface. The application developer does this by assigning a label to each domain interface member.

For step-by-step procedures for creating domain interface methods, see Defining Domain Interface Methods in the CA Aion BRE online help:

Not all methods in an Aion knowledge base can be made a domain interface member. In the following section, see Domain Interface Member Restrictions for restrictions on selected methods for domain interface membership.

Domain Interface Member Restrictions

Domain interface members can be class methods, instance methods or `_Interface` methods. The following restrictions apply to all domain interface members:

- Domain interface members must be *public* methods.
- Domain interface members used for rule Conditions cannot have any arguments; and must return a string, integer, real, or Boolean.
- Domain interface members used for rule Actions can have at most one argument, which must be either a string, integer, real, or Boolean. Any return value is ignored.
- If a domain interface member is specialized:
 - The specialization must have the same signature as the domain interface member.
 - The specialization cannot be declared to be a domain interface member in its own right. However, the implementation of an `_Interface` method can still be declared a domain interface member in its own right with its own label.
- The domain interface label must be unique within a library or application file. However, different libraries may specify the same domain interface label and a library can include another having the same domain interface label.

Dynamic Rules

Dynamic rules are rules that are assembled and defined to the knowledge base at runtime. They are not a compiled part of the knowledge-base executable. Dynamic rules are not defined in Aion BRE application source code as are non-dynamic (or *static*) rules. Dynamic rules are, in fact, complex objects constructed at runtime that the inference engine can process as if they were (static) rules. In other words, although dynamic rules are fundamentally different from static rules, they behave in the same way as static rules at runtime.

CA Aion BRE provides support for two types of dynamic rules:

- Persistent dynamic rules are rules stored in an external medium (a database) and loaded during knowledge base execution as if it were data. The external medium for storing persistent dynamic rules is called a *rulebase*. The greatest majority of dynamic rules used in Aion and considered in this documentation will be persistent dynamic rules.
- Non-persistent dynamic rules are dynamic rules created during application execution by logic internal to the knowledge base. It is expected that non-persistent dynamic rules will be used only in advanced Aion BRE applications. For more information of constructing non-persistent dynamic rules, see the chapter “Constructing Non-Persistent Dynamic Rules” in the *CA Aion BRE Rules Guide*.

CA Aion BRE dynamic rule capability currently supports only Decision Tables. Decision tables provide a high-level representation of knowledge and are easily accessible to domain experts. It should be noted that Aion IFRULEs can be represented as decision tables and therefore are amenable to treatment as dynamic rules.

Form more information on these items, see the “Decision Tables” chapter in the *CA Aion BRE Rules Guide*.

Useages for Dynamic Rules

The application scenarios and situations in which dynamic rules are recommended include the following:

- Applications requiring frequent rule updates
Where subsets of rules are changing frequently due to government regulations or ad hoc or seasonal adjustments, changes can be isolated in a dynamic rule database and installed without rebuilding the entire Aion BRE application.
- Applications requiring continuous uptime
Where applications cannot be terminated for updating rules, updates can be performed by accessing the dynamic rule databases in response to some triggering event or user interface action.
- Applications requiring on-site rule customization
Where an application is installed at a number of remote sites, dynamic rules permit each site to maintain its own set of local rules that supplement the application's static rules.

- Applications sharing sets of rules

Where multiple applications share subsets of rules, dynamic rules permit rules to be packaged into a shareable database.

Note: To share sets of rules, the applications must be constructed with the same libraries that define the domain interface members used in those rulesets.

- Situations requiring a clean separation of roles

With dynamic rules, Aion BRE application developers and domain experts can each work in an environment best suited to their roles. The application developer works within the Aion development system and the domain expert works within the dynamic rule editor (which may be customized to the domain expert's requirements).

Because Aion BRE applications and dynamic rules are loosely connected, each player can work nearly independently of the other.

Dynamic Rules Task Flow

This section specifies how dynamic rules are supported and developed throughout this process.

The essential points of the dynamic rule task flow are:

1. Once the domain interface has been defined in the Aion BRE application (domain interface labels are applied to designated methods), the Dynamic Rulebase Administrator is used to import the domain interface into the rulebase. The Dynamic Rulebase Administrator is also used to keep the Aion BRE application's domain interface "in sync" with the rulebase. For more information, see Dynamic Rulebase Administrator in the "Maintaining the Dynamic Rulebase" chapter of the *CA Aion BRE Rules Guide*.
2. After the rulebase has had domains created and populated with domain interface members, the domain expert uses the Dynamic Rule Manager to create, modify, or delete dynamic rules stored in the rulebase. For more information, see Dynamic Rule Manager in the "Maintaining the Dynamic Rulebase" chapter of the *CA Aion BRE Rules Guide*.

3. The Dynamic Rule Manager supports editors to allow the domain expert to view, create, and modify dynamic rules. Currently, the Dynamic Rule Manager supports a decision table editor for dynamic decision tables; see Dynamic Decision Table Editor in the “Maintaining the Dynamic Rulebase” chapter of the *CA Aion BRE Rules Guide*.
4. Finally, the application developer imports the DynRDLib library into the Aion BRE application. DynRDLib provides facilities to load and post dynamic rules. For more information, see the “DynRDLib” chapter in the CA Aion BRE online help. This phase of the dynamic rule task flow is described in Dynamic Rule Runtime Considerations in the “Runtime Issues” chapter of the *CA Aion BRE Rules Guide*.

A business process for maintaining dynamic rules is supported by the Dynamic Rule Repository, which is accessed through the Dynamic Rule Manager.

More Information:

[Role of the Domain Interface in System Development](#) (see page 288)

[Dynamic Rule Management](#) (see page 303)

Support for Dynamic Rules: Aion BRE-Supplied Libraries

All dynamic rules (both persistent and non-persistent) interact with the executing Aion BRE application using one or more Aion-supplied libraries:

- DynRLib provides facilities for defining and posting dynamic rules. For more information, see the “DynRLib” chapter in the CA Aion BRE online help.
- DynRDLib (which includes DynRLib) provides facilities for loading dynamic rules from the rulebase. The library specializes the Aion Query class, permitting dynamic rule editors and Aion BRE applications to use selection markers corresponding to columns in the database tables. DynRDLib only allows loading rules from the rulebase and does not support updating the rules, domains, or definitions of the domain interface members. For more information, see the “DynRDLib” chapter in the CA Aion BRE online help.
- DynRELib (which includes DynRDLib and DynRLib) provides facilities updating the rulebase. It can be used to develop a custom dynamic rule editor or to develop utilities to maintain domain or domain interface information stored in the rulebase. For more information, see the “DynRELib” chapter in the CA Aion BRE online help.

Note: See the Aion Examples folder for sample Aion BRE applications that illustrate the use of the library services. It is recommended that the Aion developer explore the methods contained in DynRLib and DynRDLib to learn the capabilities of these libraries and to find the useful methods that they offer. The MDDRules example illustrates advanced uses of DynRLib and DynRDLib operations.

External Rules: Use Dynamic Rules or Generate Static Rules with COBSLib?

Persistence has come to be an industry standard term referring to the ability of a program construct such as a class or instance to continue to exist in an external medium after the program executable has closed down. Rule persistence is also referred to as *rule independence*, and persistent rules are often called *external rules*.

Domain interface technology is one way that CA Aion BRE allows rules to be externalized outside applications. Another strategy for maintaining external (persistent) rules uses the powerful capabilities of COBSLib. COBSLib is an Aion library that contains the classes and methods necessary to create, browse, modify, build, and even run an Aion BRE application. It can be considered an API on top of the building services of Aion. For more information on COBSLib, see the "COBSLib" chapter in the CA Aion BRE online help. A COBSLib approach to externalizing rules allows persistent rules to be generated *as static rules* in an Aion BRE application. This section compares the dynamic rule and COBSLib approaches for externalizing rules.

How does the COBSLib approach work? With this approach, the Aion BRE application developer writes an Aion knowledge base that includes COBSLib. This knowledge base can use Aion's database connectivity and Query classes to access the external rules that the domain expert has saved in a database. It can then use COBSLib functionality to open an existing Aion BRE application and to generate rule methods and literally write the rules for that application. The logic in the COBSLib application is to format the database structures in which the rules are stored into the syntax of Aion rules and insert these reformatted rules into the appropriate rule methods in the target Aion BRE application as static rules. The COBSLib application has the "knowledge" to perform these activities. The target application can then be compiled, if you wish, by the COBSLib application itself.

CA Aion BRE provides the base libraries (COBSLib, WinLib, DataLib) to enable the Aion BRE application developer to write such a knowledge base. However, unlike what it provides for dynamic rules, Aion does not provide the structure in which persistent rules are stored or any external rule editors intended for non-programmers as part of its COBSLib approach. Thus, there are both advantages and disadvantages to using the COBSLib approach instead of the dynamic rule approach.

There are two principal advantages of using the COBSLib approach:

- The COBSLib approach can accommodate rules of any format.
- The COBSLib approach does not incur any performance degradation.

Accommodation of Rules for Any Format

Currently, dynamic rules are limited to decision tables. While it is anticipated that Aion will offer additional types of dynamic rules, dynamic rules will be always be restricted to a subset of rule types. In the COBSLib approach, the Aion BRE application developer can select any type of Aion rule and design a database structure to support that type. For instance, an IFRULE database structure could easily be distinguished from (a very similar) WHEN structure so that the Aion COBSLib application would know what production rules (as opposed to WHEN demons) to generate for the target application.

No Performance Degradation

When dynamic rules are posted to the inference engine they must undergo extensive validation and interpretation. After all, the executing knowledge base does not know whether the particular rules it loads have been tested. Perhaps the rulebase and knowledge base are out of sync! There will be a performance penalty for this validation. Because the COBSLib approach generates static rules, there is no performance penalty. At runtime, COBSLib-generated static rules are no different than rules written by the application developer.

The advantages of dynamic rules over the COBSLib approach include:

- Immediate use of external rules.
- Runtime loading of rules.

Immediate Use of External Rules

Aion offers the opportunity to begin using dynamic rules immediately upon installing Aion. Aion provides the database (rulebase) in which to store external rules, the utility (Dynamic Rulebase Administrator) to synchronize this database with the Aion BRE application, and basic editing capabilities for persistent dynamic rules in the Dynamic Rule Manager. By contrast, these facilities must be designed and developed by the Aion BRE application developer when using the COBSLib approach. In particular, with the COBSLib approach, a data model for storing rules must be designed, an appropriate editor must be developed, and, most importantly, an additional Aion BRE application using COBSLib must be written.

Runtime Loading of Rules

There are many business applications that directly benefit from runtime loading of rules. For examples of such applications, see "Usage for Dynamic Rules" in the Dynamic Rules chapter. By contrast, the COBSLib approach still involves a compilation step in order to put rule changes into effect. Indeed, compilation becomes a job stream consisting of first executing the Aion COBSLib application and then compiling the target knowledge base. The major advantage of dynamic rules is that they do not require recompilation in order to effect rule changes. Obviously, however, the tradeoff in achieving runtime loading of rules is the performance penalty that the Aion knowledge base incurs when validating dynamic rules.

Which approach is best for you depends upon the requirements of your application:

- Are your rules amenable to the formats provided by Aion's dynamic rule capabilities?
- What is the allowable interval between recognizing a rule change and needing it implemented?
- What are the performance requirements for the Aion BRE application itself?

Whichever approach is adopted, adequate planning and appropriate resources are required to make any project successful.

Styles of Rules

External rules must be distinguished from internal rules that are explicitly defined in the Aion BRE language. This distinction must be contrasted with the very different distinction between static and dynamic rules. Static rules are those that are compiled as part of the executable; dynamic rules are defined to the knowledge base at runtime. The differences among these concepts yield the following matrix of the styles of rules supported by Aion BRE.

Static	Dynamic
---------------	----------------

External/ Persistent	Aion + COBSLib	Persistent dynamic rules (supported by a domain interface to an external physical medium)
Internal	Tradition Aion Rules (IFRULEs, Pattern Matching rules, Demons)	Non-persistent dynamic rules

In general, internal static rules will be the most common rules used by the Aion BRE application developer. Non-persistent dynamic rules will probably find a place only in advanced Aion BRE applications. For uses of non-persistent dynamic rules, see the chapter "Constructing Non-Persistent Dynamic Rules" in the *CA Aion BRE Rules Guide*. Internal rules have the disadvantage, however, of being maintainable only by an application developer who is familiar with the Aion IDE.

External rules have the advantage of allowing maintenance outside of the Aion IDE by the domain expert. Thus, externalizing rules is often the result of an explicit system requirement. The section Externalizing Rules: Using Dynamic Rules or Generating Static Rules with COBSLib? discusses the issues involved in selecting between different strategies for externalizing rules.

Chapter 10: Use the Rule Manager Wizard

Business rules are those rules that are maintained directly by a business domain expert. This definition complies with goals of the **business rules approach** by providing the business domain expert with the means to maintain the rules of business operations. Aion BRE provides the Aion Rule Manager for external management of business rules. For information on the Aion Rule Manager, see the *CA Aion Rule Manager Product Guide*.

CA Aion BRE provides a Rule Manager Wizard, which automatically generates the Aion BRE code to communicate with a CA Aion Rule Manager rulebase that has been deployed as a Web service.

This section contains the following topics:

[Process Overview](#) (see page 299)

[Invoke the Rule Manager Wizard](#) (see page 300)

Process Overview

Using the Rule Manager Wizard presumes that you have created a Rule Manager project in the environment and also deployed it as a rulebase using the Rules Application Server (RAS). For more information about these tasks, see the

The general procedure is as follows:

1. Create your Aion BRE application in the IDE.
2. Invoke the Rule Manager Wizard, which reads the Web service definition language (WSDL) of the desired rulebase and generates three classes which can be used to call the Rule Manager Web service.
3. Write the Aion BRE code to use these three classes to invoke the rules in the rulebase.
4. Test the resulting Aion BRE application.

Invoke the Rule Manager Wizard

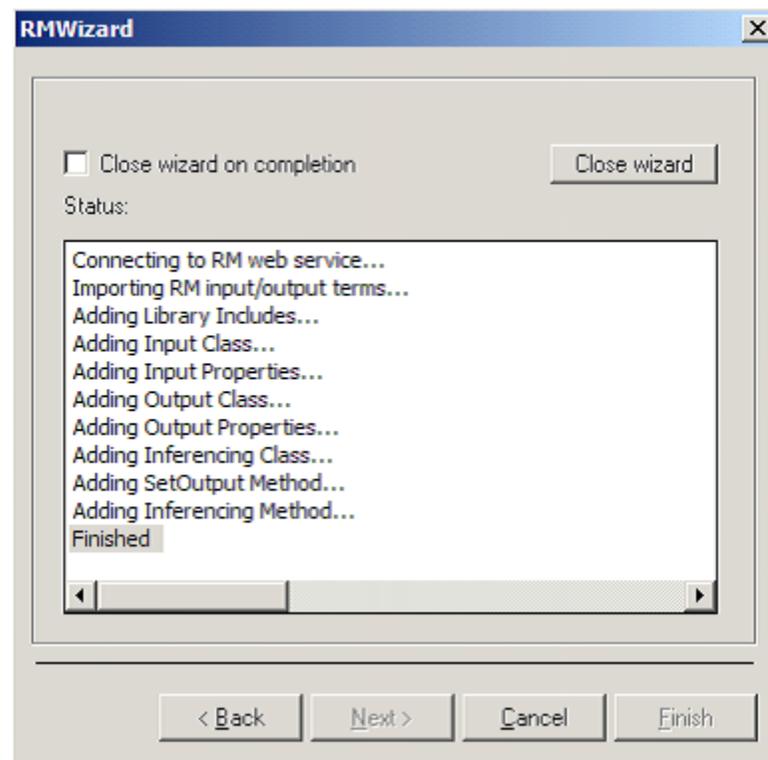
This procedure invokes the Rule Manager Wizard.

To invoke the Rule Manager Wizard, follow these steps:

1. Select Tools, Rule Manager Wizard...
2. The first Rule Manager Wizard dialog appears.

Enter the machine name and port of the server where the Rule Manager Web service is running. The machine name can be localhost if the Web service is running on the same machine as CA Aion BRE. The port number is typically 8080 unless it was changed during Portal installation.

3. Click Next.
4. A list of deployed rulebases displays.
5. Select the deployed rulebase you want.
6. Click Next.
7. A processing status dialog displays.
8. If you want to retain status messages after processing occurs, uncheck the check box Close wizard on completion.
9. Click Finish to start Web service processing.



When Web service processing is complete, you should see three new classes in your application:

```
wsname _Input  
wsname _Output  
wsname
```

where *wsname* is the name of the Rule Manager Web service.

Example:

These classes are available for use in your CA Aion BRE application. Here is an example of how they are used to invoke the rules in the Rule Manager Web Service.

```
// Declare variables for the classes  
var AionInput is & wsname_Input  
var AionOutput is & wsname_Output  
var rules is & wsname  
  
// Create instances of the classes  
AionInput = wsname_Input.create  
AionOutput = wsname_Output.create  
rules = wsname.create  
  
// Set the input values  
AionInput.setFieldName1 (value)  
AionInput.setFieldName2 (value)  
AionInput.setFieldName3 (value)  
  
// Set the trace level  
AionInput.setTraceLevel(0)  
  
// Invoke the rules  
rules.inferRM(AionInput,AionOutput)  
  
// Retrieve the results  
messagebox(AionOutput.getFieldNameN)
```


Chapter 11: Dynamic Rule Management

CA Aion BRE supports dynamic rules. Dynamic rules allow the business expert to maintain Aion rules outside of the Aion development environment using the Dynamic Rule Manager. For more information on maintaining dynamic rules and the Dynamic Rule Manager, see Dynamic Rule Manager in the “Maintaining the Dynamic Rulebase” chapter in the *CA Aion BRE Rules Guide*.

Note: Currently, decision tables are the only supported form of dynamic rules.

CA Aion BRE also extends the dynamic rule management functionality by providing support for a business rule management **process**. A business process for administering rule management is necessary as business users take over responsibility for maintaining their business rules. This is especially true if it is expected that business users will maintain actual production rules in order to take advantage of CA Aion BRE's ability to load dynamic rules at runtime without recompilation of the application. Although the end-user business community must ultimately be responsible for defining the process of managing rules, the rule management software should support essential functionality that would be found in any such process.

Aion provides this support in two ways:

- User access permissions can be controlled at a user / domain level.
- The Dynamic Rule Manager is extended with access to a rule repository for managing rule maintenance.

The rulebase that is provided with CA Aion BRE is a database for storing rules. A rule repository should also provide rule check-out and check-in services while maintaining prior versions of rules. To meet these objectives Aion uses a connection to an Microsoft Source Code Control (SCC) program. With this approach, there is no impact on the size of the dynamic rulebase itself to maintain prior versions of rules.

The features of CA Aion BRE that support business rule management are presented in The Business Rule Management Process.

This section contains the following topics:

[Rule Repository Functionality](#) (see page 304)

[The Business Rule Management Process](#) (see page 305)

Rule Repository Functionality

CA Aion BRE implements the rule repository extension to the rulebase by saving rule versions as **text** in a source code control system.

Rule Repository functionality requires software compliant with the Microsoft Source Code Control (SCC) standard. Thus, any SCC compatible source code repository can be used with the Aion Dynamic Rule Manager. Typical SCC compatible repositories include:

- Microsoft's Visual SourceSafe
- CA Software Change Manager

Note: CA Aion BRE does not provide a source code control system.

Rules must be in the form of a text file (.txt) to be stored in the Repository. When adding a rule to the rule repository, the Dynamic Rule Manager converts the rule into a stream of character data and writes this data to a temporary text file. This text file is passed into the source code control system for storage in its repository. The opposite occurs when a rule is retrieved from the repository into the Manager.

The Dynamic Rule Manager handles the conversion and storage of rules between the source code control system and physical rulebase invisibly to the business user.

Note: CA Aion BRE installation creates a default directory for storing the temporary files used in the conversion between the rule repository and the rulebase. There should be no reason to change this default directory.

Set Up the Rule Repository

Follow installation instructions for your choice of source code control program, and establish a network accessible common source code database as your rule repository.

During initial set-up, the structure of the rule repository should be defined in the source code control database. A typical structure would be to define a project of Rule Repositories (to include all dynamic rules) and subprojects to define separate rulebases. Each rulebase should contain project folders that reflect the Domains within the production dynamic rulebase.

Note: While it is not a requirement, it is helpful if the names of rulebase projects match exactly to the domain names in the rulebase.

A typical structure in the rule repository might be:

```
+ Aion Rule Repositories
|
|--  + Sales Rulebase
|    |
|    |--  Product Domain
|    |
|    |--  Inventory Domain
|
|--  + Employee Rulebase
|    |
|    |--  Compensation Domain
|    |
|    |--  Benefits Domain
```

The structure within the source code control program database should reflect, but it is no limited to, the structure of the production rulebase. For example, structures to store test versions of rules prior to productionalizing them could be defined.

Procedures should be defined within a company for how the dynamic rule repository is to be used. It is important to remember that source code control and update control of the rulebase are two different issues. Because the Aion rulebase is a database, update control must be exercised by standard database security measures. It may be desirable to have a single point, a Rulebase Administrator, that controls creating rules in the production rulebase.

The rule repository may be used as a staging area from which rules composed by business experts are moved by the Rulebase Administrator to the rulebase for testing and productionalizing. It may be desirable to define both test and production domains in the rulebase.

The Business Rule Management Process

This section describes features of CA Aion BRE that support a business rule management process. Aion addresses two aspects of this process:

- Establishing user access levels in order to allow/prohibit users to exercise specific capabilities of the rule manager, and
- Managing rule maintenance through check-out/check-in and other administrative services.

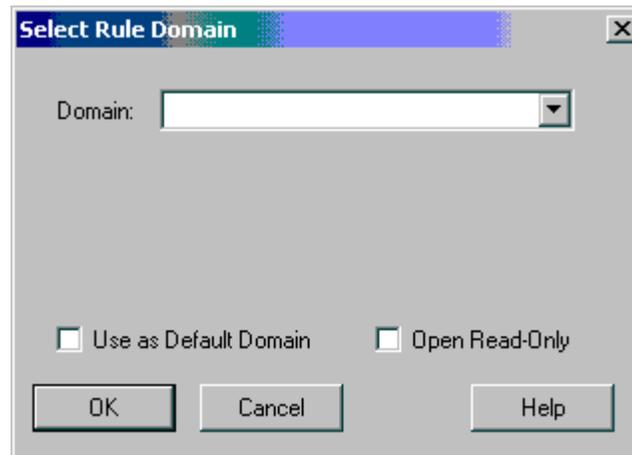
Establish User Access Permissions

The first step in a business rule management process is typically the administration of who can access the dynamic rulebase and of the privileges that these users can exercise when accessing the rulebase. In Aion BRE, the functionality to establish user access levels is provided in the Dynamic Rulebase Administrator.

User access permissions are defined at the user/domain level. In other words, each user may be given specific access privileges for each domain in the rulebase. Currently, Aion BRE provides three levels of user access: Prohibited, Read-only, and Read-Write. For a step-by-step description of establishing user access permissions, see Establishing User Permissions in the CA Aion BRE online help.

User Permissions are automatically read by the Dynamic Rule Manager. The UserID associated with the levels of permissions defined in the Administrator must match the UserID that is defined for the rulebase connection (not the User Name) in the Dynamic Rule Manager. For more information on defining the rulebase connection in the Dynamic Rule Manager, see "Selecting a Rulebase and Opening a Domain" in the *CA BRE Aion Rules Guide*. (Although a UserID is not normally required for an ODBC connection to a rulebase under Microsoft Access, a UserID would be required on the connection definition if permissions are active in the rulebase.)

Note: The Dynamic rule Manager now provides an option to open a domain as read-only in the Open Domain, Select Rule Domain dialog:



The Open Read-Only box is automatically checked if the user has Read-Only access privileges.

Dynamic Rule Environment Controls

The issues of rulebase update security, user level access permissions to the Dynamic Rule Manager, and rule repository access are separate issues. The following table summarizes the different controls that are available:

Source	Maintained through	Access Services
Rulebase Security	Database manager and database security rules	Ability update, change, delete rules within the physical rulebase.
Dynamic Rule Manager	Dynamic Rule Administrator	Ability to use facilities of the Dynamic Rule Manager. See following Note.
Dynamic Rule Repository	Source Code Control System	Ability to read and write to the source code control database.

Note: It is possible that the Rulebase Security and Dynamic Rule Manager may have inconsistent access permissions for the same user. For example, the rulebase may allow a user only read access to the rulebase while the Dynamic Rule Manager may permit the user read-write access. In these cases, the more restrictive rule will take precedence.

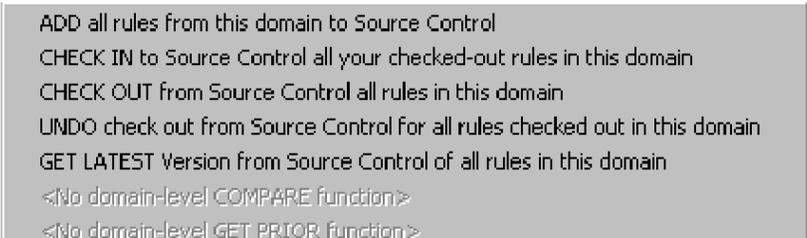
Dynamic Rule Repository Functionality

The dynamic rule repository functionality is accessed through the Dynamic Rule Manager. The Dynamic Rule Manager allows the user to access single rules in the repository or to access all rules in the currently active domain. Parallel functionality is provided for both single rules and all rules in a domain, with only two exceptions.

Note: Each repository request during a session will invoke the source control program login dialog. The Dynamic Rule Manager user should login to rule repository. Other source code control program dialogs will be invoked as necessary while working within the repository.

Access Single Rules

To access rules on a single rule basis, select and right click on a rule name in the domain workspace. This action brings up a pop-up menu that exposes the available functions of the source code control system for single rules. (Options that are not available for the highlighted rule will be grayed out.)

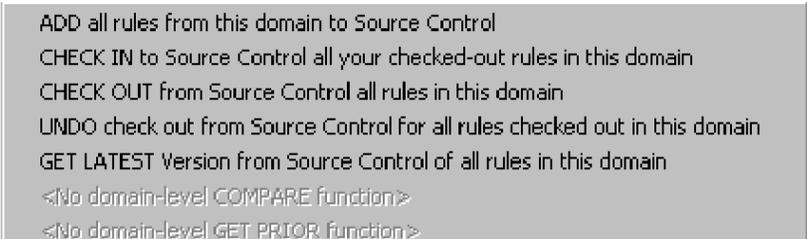


```
ADD all rules from this domain to Source Control
CHECK IN to Source Control all your checked-out rules in this domain
CHECK OUT from Source Control all rules in this domain
UNDO check out from Source Control for all rules checked out in this domain
GET LATEST Version from Source Control of all rules in this domain
<No domain-level COMPARE function>
<No domain-level GET PRIOR function>
```

Selection of any menu item requires that the selected rule **not** be displayed in an open Rule editor.

Access All Rules in a Domain

To access all rules from the currently active domain, right click on any area of white space in the domain workspace. This action brings up a pop-up menu that exposes the available functions for accessing all rules in the domain.



```
ADD all rules from this domain to Source Control
CHECK IN to Source Control all your checked-out rules in this domain
CHECK OUT from Source Control all rules in this domain
UNDO check out from Source Control for all rules checked out in this domain
GET LATEST Version from Source Control of all rules in this domain
<No domain-level COMPARE function>
<No domain-level GET PRIOR function>
```

Note: Options to compare a rule to one in the rule repository and to get a prior version of a rule are not available when accessing all rules in a domain. All other options are available if appropriate rules exist on which to perform the option (for example, CHECK IN is not available if no rules are currently checked out to the user).

Selection of any menu item requires that all editors be closed.

Add Rules to the Repository

To add dynamic rules to the rule repository, select the Add option from either the Single Rule or Entire Domain pop-up menu. Selecting Add from the Single Rule menu will add the currently selected rule to the rule repository; selecting Add all rules from the Entire Domain menu will add all rules in the current domain to the repository. All the original copies of the rules will remain present in the rulebase.

It is possible to add a rule that is currently in a project in the rule repository to another repository project. Aion BRE will detect this situation and change the confirmation prompts accordingly. For example, when all rules in a domain are added to the rule repository, a preliminary prompt will confirm whether rules already in the rule repository should be included (if there are any such rules in the set to be added). Answer this prompt Yes only if you wish to add the rules to a different project than they are currently in.

When a rule has been added to the rule repository, it is flagged in the Domain Workspace with "[in SC]." A rule that is in the rule repository may be checked-out for maintenance.

More Informaton:

[Access Single Rules](#) (see page 308)

[Access All Rules in a Domain](#) (see page 308)

[Rule Check Out and Check In](#) (see page 309)

Rule Check Out and Check In

Once a rule is added to the rule repository, it is available for check-out. Rules available for check out are marked in the workspace with the flag "[in SC]". To check-out rules select Check Out from either the Single Rule or Entire Domain pop-up menu; for more information on these pop-up menus. Selecting Check Out from the Single Rule menu will check-out the currently selected rule; selecting Check Out from the Entire Domain menu will check out all rules for the current domain that are in the repository.

Note: If any rule is checked out of a domain to any user, check out of the entire domain is not available; in this case each desired rule must be checked out individually.

The checked-out rule will overlay the current copy of the rule in the domain of the rulebase that is currently opened in the Dynamic Rule Manager. This domain may be in the production rulebase or in a rulebase on the local machine of the business rule expert.

When a rule is checked-out, it is flagged in the Domain Workspace with [out SC to *userID* on *timestamp*]. (The repository uses the connection user ID, as specified in Settings.User ID (not User Name), to identify who owns a checked-out rule.) No other user of the rule repository can modify the source code of the rule in the repository except for the holder of the check-out.

Important! The issues of rulebase update privileges, user level access privileges (to the Dynamic Rule Manager), and rule repository access are different issues. See Dynamic Rule Environment Controls for a summary of these issues. In particular, a rule whose source code is checked out of the rule repository may still be changed in the rulebase. It is therefore important to have a central control on rulebase updating, such as a rulebase administrator.

Note: Checking out rules at the domain level will check out only those in the repository that reside in the current rulebase.

When a rule is checked-out of the repository, the following options are activated on the rule repository pop-up menu only for the current holder of the check out lock.

Undo Check Out

Cancels the check out lock in the rule repository. In effect, this option says that the holder of the check-out wants to disregard any locally made changes to the rule and retain the current version. Selecting Undo Check Out on the Single Rule menu undoes the check out only on the currently selected rule; selecting Undo Check Out on the Entire Domain menu undoes the check out on all rules currently checked out by the user for the current domain. This action restores the [in SC] status to the rule.

Check In

Replaces the rule in repository with a new copy of the rule from the rulebase that was maintained by the holder of the lock. Selecting Check In from the Single Rule menu replaces the currently selected rule; selecting Check In from the Entire Domain menu replaces all rules currently checked out by the user for the current domain. Check In restores the [in SC] status to the rule.

More Information:

[Business Rule Maintenance Scenarios](#) (see page 312)

[Access Single Rules](#) (see page 308)

[Access All Rules in a Domain](#) (see page 308)

[Dynamic Rule Environment Controls](#) (see page 307)

Rule Versioning

Because the Dynamic Rule Manager accesses the API of a source code control system, versioning is automatically provided. Versioning, or the storage and retrieval of prior version of rules, provides the following functionality that is helpful in supporting a rule management process:

- Get latest version of a rule.

To get the latest version of a rule, select Get Latest from either the Single Rule or Entire Domain pop-up menu. Selecting Get Latest from the Single Rule menu retrieves a copy of the latest version of the selected rule; selecting Get Latest from the Entire Domain menu retrieves all rules in the current domain from the rule repository.

Getting the latest version of the rule replaces the copy of the rule in the current rulebase with a copy of the latest version from the repository. The copy is flagged to indicate current status in repository. Retrieved rules are writable, but they cannot be checked back into the repository.

Note: Getting the latest versions of rules at the domain level will retrieve only those rules in the repository that reside in the current rulebase. Rules are retrieved regardless of their check-out status, which remains unchanged.

- Get prior version of a rule.

To retrieve a prior version of a rule, select Get Prior from the Single Rule pop-up menu.

Note: Get Prior is not available on the Entire Domain pop-up menu.

Selecting Get Prior will invoke the Get Historical functionality offered by the source code control system for retrieving a prior version of the currently selected rule. The retrieved rule is flagged to indicate its current status in the repository. The retrieved rule is writable, but it cannot be checked back into the repository.

- Compare two versions of the same rule.

To compare the current rule in the rulebase with the latest version of that rule contained in the repository, select Compare from the Single Rule pop-up menu.

Note: Compare is not available on the Entire Domain pop-up menu.

This option will open the Compare window in the Dynamic Rule Manager, which presents read-only copies of the selected rule (from the rulebase) and of the rule from the repository.

More Information:

[Access Single Rules](#) (see page 308)

[Access All Rules in a Domain](#) (see page 308)

Summary: Effect of Rule Repository Functions on the Rule in the Current Rulebase

The following effects occur with respect to the contents of the current rulebase for each of the rule repository functions.

Note: The current rulebase may be either the production rulebase or a local rulebase on the business expert's PC.

Add, Check in

The local rule is flagged to indicate that it is currently in source code control system.

Check out

The local rule is replaced with the latest version from rule repository, and is flagged as checked out to the current user ID.

Undo

The checked out flag and user ID are removed from local rule.

Get latest, get prior

The local rule is replaced with a version from the rule repository and is flagged to indicate its current status in the repository.

Compare

There is no effect.

Business Rule Maintenance Scenarios

This section describes several scenarios for using the dynamic rule repository functionality in a business rule management process.

Create New Rules in the Rulebase

Dynamic rules that are added to the rulebase do not automatically get added to the rule repository. Therefore, there must be a business process for insuring that rules are properly added to the repository. Here we assume that the business expert is working from a local rulebase when creating a rule. The rulebase administrator controls update to the production rulebase.

1. In the Dynamic Rule Manager, select New, Decision Table from the Rule menu, or click the New Decision Table button.
2. Complete the definition of the rule in the local rulebase. Save the rule. (Its status should be Inactive; it may be saved and added to the repository even if it is invalid.)
3. Right click the rule name in the Domain Workspace. Select "Add" from the pop-up menu.
4. Log into the source code control system (rule repository) and select the appropriate project folder into which the rule should be added.

At this point the Rulebase Administrator could be informed that a new rule is ready to be added to the rulebase. The Rulebase Administrator could then use the Get Latest function to retrieve the latest version of the rule for repopulating the production rulebase.

Change Dynamic Rules

Changing a production dynamic rule requires the rule to be checked out of the rule repository.

1. Check out the rule to be changed to a test domain in the rulebase or to a local rulebase.
2. Change the test/local copy of the rule and check it back into the rule repository.
3. Inform the Rulebase Administrator that a rule is ready to be modified in the rulebase.
4. The Rulebase Administrator gets the latest version of the rule into the production rulebase. This action will delete the existing rule and create the modified rule in its place.

Production of Dynamic Rules

The business rules process is likely to involve a process of testing the rules to be added to the production rulebase. In this case, the Rulebase Administrator may get the latest version the rule into a test domain of the rulebase. The Aion BRE application can then be run from the test domain.

Chapter 12: Aion BRE Reports

You can generate reports from within your Aion BRE applications. Report output can be generated to a text or HTML file, it can be displayed in a window, and/or it can be sent to a printer.

CA Aion BRE provides two libraries to assist in creating reports: IOLib and IOWLib. These libraries contain methods and attributes to use in designing and implementing reports.

This section contains the following topics:

[About Aion BRE Reports](#) (see page 315)

[Aion--IOLib Classes](#) (see page 317)

[IOWLib Classes](#) (see page 319)

[Work with a Report Canvas](#) (see page 319)

[How You Use the Artists](#) (see page 325)

[Sample Application](#) (see page 327)

[The Sample Canvas](#) (see page 328)

[Output Options](#) (see page 329)

About Aion BRE Reports

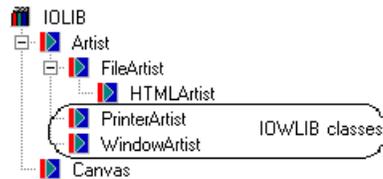
In CA Aion BRE, reports are created by combining a canvas and an artist.

The canvas contains the specifications for the layout and content of the report. Different parts of an application, such as rules and methods, can write to the canvas as appropriate during application processing.

The artist renders a specified canvas to a particular output device, such as a file, a printer, or a window in the user interface. Different kinds of artists, represented by different classes in Aion system libraries, know how to render canvases to different devices. Different artists can render the same canvas to different devices.

Aion--IOLib and IOWLib

IOLib and IOWLib are actually a single class hierarchy that is divided into two libraries:



- IOLib contains, in its FileArtist and HTMLArtist classes, methods for generating a report to a text file or an HTML file. It also contains the Canvas class, as well as a number of other classes, whose methods you do not normally need to call directly.
- IOWLib contains, in its PrinterArtist and WindowArtist classes, methods for generating a report to a printer or window.

The IOWLib library includes the IOLib and WinLib system libraries, so it supports the rendering of reports to all of the possible output devices. Because of this organization, you can include just IOLib when generating HTML or text file reports, without incurring the overhead of including WinLib. Because IOWLib contains IOLib, it is never necessary to explicitly include both libraries in a custom library.

Aion--How you Create a Report

Typically, implementing a report in an Aion BRE application includes these steps:

- Include IOWLib or IOLib in your application, depending on how you want to output the report. This gives you access to the classes containing report-related methods and attributes.
- If the report will be returning data from a database, define a connection to be associated with that database.
- If outputting to a window:
 - Create and design the AppWindow to contain the report.
 - Choose or generate an EditWindow class to contain the report-related methods.

- Design the physical layout of the report, and determine any static information, headers, and footers.
- At runtime, create a dynamic instance of the Canvas class, and call its methods to implement the design and to add content to the report.
- At runtime, create a dynamic instance of the appropriate Artist class or classes, then call Artist methods to render output based on the Canvas instance.

Aion--IOLib Classes

This section gives a general overview of the classes contained in IOLib. See the CA Aion BRE online help for more information about the methods for these and other Aion BRE classes.

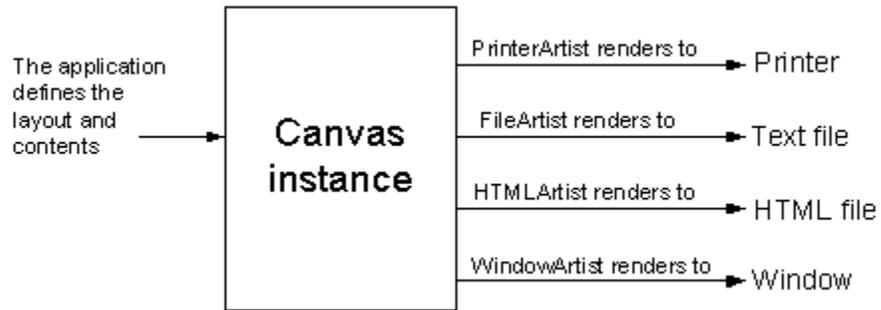
Canvas

The methods in the Canvas class are used to specify the report layout. Use them to specify the following information about your report:

- Font attributes
- Background color
- Background image
- Footer text and positioning
- Header text and positioning
- Page width and pagination
- Table definitions (including headers and column widths)
- List definitions (including numbered and bulleted lists)

Artist

IOLib contains the Artist base class. Both IOLib and IOWLib contain additional classes derived from Artist that are specific to the output type and device. Reports for text file, HTML file, printer, and window output use the classes FileArtist, HTMLArtist, PrinterArtist, and WindowArtist, respectively.



To output the report to a specific device (file, window, or printer), you can use the methods in each respective Artist (FileArtist, WindowArtist, and so on). Use the methods in the derived Artist classes to perform the following actions:

- Set attributes of the report that are specific to the device
- Render a canvas to the predefined device

Fine Artist

The FileArtist class contains methods for report generation to plain text files; in particular, for specifying when, how, and where file output is sent.

Use methods in FileArtist to do the following:

- Specify which text file to render to
- Render the supplied canvas to the specified file

HTMLArtist

The HTMLArtist class contains methods to generate files in HTML, in particular, for specifying when, how, and where HTML file output is sent.

Use methods in HTMLArtist to do the following:

- Specify which file to render HTML output to
- Render the supplied canvas to the specified file

IOWLib Classes

This section gives a general overview of the classes contained in IOWLib. See the CA Aion BRE online help for more information about the methods for these and other Aion BRE classes.

WindowArtist

WindowArtist provides methods for rendering a canvas to a window.

Use WindowArtist methods to perform these operations on a defined canvas:

- Specify an EditWindow to render the output to
- Render the supplied canvas to the specified EditWindow

PrinterArtist

PrinterArtist provides methods for rendering a canvas to a printer.

Use the methods in PrinterArtist to perform these operations on a defined canvas:

- Specify a printer to render the output to
- Query the user for which printer to render to
- Render the supplied canvas to the specified printer

Work with a Report Canvas

Working with a report canvas can involve the following general steps:

- Creating the Canvas Instance
- Starting the Report Page
- Defining the Overall Appearance
- Specifying Font Attributes
- Writing Text and Images to the Canvas
- Adding Blank Lines
- Specifying Tables
- Ending the Report

Create the Canvas Instance

Call the `Canvas.Create()` method to create a dynamic instance of `Canvas`. It is useful to set a pointer to the value that the `Create()` method returns

Example:

```
pReport = Canvas.Create( )
```

Typically, the report canvas is created in a separate method of type pointer to `Canvas`.

Start the Report Page

A canvas always contains at least one page. Start the first page of the canvas using the `canvas.StartPage()` method, where `canvas` is a pointer to a dynamic canvas instance.

Example:

```
pReport.StartPage( )
```

`StartPage()` takes several optional arguments. The first argument, `width`, is an integer specifying the requested number of characters in each row of the report. For devices without an inherent width, this number defaults to 80. There are other optional arguments that are generally used only by `HTMLArtist`.

Define the Overall Appearance

Use the `Canvas.Set*` methods to optionally define any or all of the following:

- Use `canvas.SetName(string)` to specify a report title. The string argument is a quoted string. This is used as the value of `FMT_NAME` in headers and footers and as the Document Name by the `HTMLArtist`.

Example:

```
pReport.SetName("Monthly Sales Report")
```

- Use `canvas.SetHeader(left-string, center-string, right-string)` and/or `canvas.SetFooter(left-string, center-string, right-string)` to specify the text and alignment for the header and/or footer. The arguments are strings specifying the header and footer text:
 - *left-string* specifies text to be left-aligned
 - *center-string* specifies text to be centered
 - *right-string* specifies text to be right-aligned

Any of these arguments can be NULL.

Example:

This example specifies that the header has left-aligned text including the date and that the footer has a right-aligned page number:

```
pReport.SetHeader("Sales Data for " & FMT_DATE)
pReport.SetFooter(NULL, NULL, FMT_PAGE)
```

Note: Use the `FMT_*` constants in the Canvas class, such as date (`FMT_DATE`) or page number (`FMT_PAGE`), to add automatically updated elements to the header.

- Use `canvas.SetBackgroundColor(color)` to specify a color for the report background. The color argument can be a string containing the color name (in quotation marks), or in hexadecimal format ("`#RRGGBB`").

Example:

```
pReport.SetBackgroundColor("cyan")
pReport.SetBackgroundColor("#00A5C6")
```

Note: Only `WindowArtist` and `HTMLArtist` use the `SetBackgroundColor()` method. The color name strings that you can use in an HTML file depend on the browser used to view the file. The color name strings you can legally use with `WindowArtist.SetBackgroundColor()` are specified in the `WindowArtist.StartPage()` method.

- Use `canvas.SetBackgroundImage(image)` to specify an image file to display in the background of an HTML report. The image argument is an image filename, enclosed in quotation marks.

Example:

```
pReport.SetBackgroundImage("CompanyLogo.jpg")
```

Note: The `SetBackgroundImage()` method is only valid for canvases that are rendered using `HTMLArtist`.

Specify Font Attributes

Use the canvas methods `FontBold()`, `FontNormal()`, or `FontFixed()` to specify whether to use a Bold, Normal, or Fixed pitch font, respectively.

Example:

```
pReport.FontBold( )
```

Write Text and Images to the Canvas

- Use `canvas.Write(string)` to write a specified string to the canvas and to advance to the next paragraph. The *string* argument is a quoted string. For example:

```
pReport.Write("The information in this report is CONFIDENTIAL.")
```

- Use `canvas.WriteNative(string)` to add an uninterpreted string to the output. The *string* argument is a quoted string.

Example:

```
pReport.WriteNative("This report is FOR YOUR EYES ONLY.")
```

Note: When rendering a canvas, all derived Artist classes except `HTMLArtist` treat strings added to the canvas using `Write()` and `WriteNative()` identically. `HTMLArtist` forces strings added with `Write()` to an HTML paragraph format; it does not do this to strings added using `WriteNative()`.

- Use `canvas.WriteCentered(string)` to write a specified string to the canvas, centered on the page, and then proceed to the next paragraph. The string argument is a quoted string.

Example:

```
pReport.WriteCentered(format(CurrentDate) & "Report")
```

- Use `canvas.WriteList(list, ordered)` to format and add each list object to the canvas.
 - The list argument is a list of strings.
 - If the Boolean `ordered` argument is TRUE, this list is numbered. In `ordered` is FALSE or not specified, the list is bulleted.

Note: You can use the Boolean constant `FMT_NUMBERED` as the `ordered` argument in place of TRUE to make your code clearer.

Example:

The following example writes a numbered list with three elements to the canvas represented by `pReport`:

```
pReport.WriteList(List("Eat breakfast", "Eat lunch", "Eat dinner"),  
FMT_NUMBERED)
```

- Use `canvas.WriteLine()` to draw a horizontal line the width of the output device.

Example:

```
pReport.WriteLine( )
```

- Use `canvas.WriteImage(image, target)` to specify an image to display in the body of an HTML report.
 - The image argument is a string containing the name of an image file.
 - The optional target argument is a string specifying the URL for the hypertext target for the image.

Example:

```
pReport.WriteImage("CompanyLogo.jpg", "http://www.MyCompany.com")
```

Note: The `WriteImage()` method is only valid for canvases that are rendered using `HTMLArtist`.

Add Blank Lines

Use the `canvas.SkipLine(lines)` method to skip one or more lines before resuming output. The `lines` argument is an integer, representing the number of lines to skip.

Example:

```
pReport.SkipLine(3)
```

Specify Tables

You can use the following table-related methods to specify a table in the report.

- Use `canvas.StartTable(headers, width)` to begin a table definition.
 - The `headers` argument is a list of strings representing the header titles, with one string for each column. NULL is an acceptable value.
 - The `width` argument is a list of integers representing the column widths (in characters).

Both argument lists should have the same number of elements.

Example:

```
pReport.StartTable(List("Part Number", "Unit Price", "Quantity"), List(15, 15, 10))
```

Use `canvas.WriteRow(strings)` to set text for the current row in the table. The `strings` argument is a list of strings, one string for each column in the table.

Example:

```
pReport.WriteRow(List("AB-579433", "8.95", "25"))  
pReport.WriteRow(List("AF-660324", "12.45", "15"))
```

Use `canvas.EndTable()` to end a table definition.

Example:

```
pReport.EndTable()
```

End the Report

- Use the `canvas.EndPage()` method to end the canvas.

```
pReport.EndPage()
```

You can also use the `EndPage()` method followed by another `StartPage()` method to force a page break within a canvas. You might want to do this, for example, to force a table to begin at the top of a new page or to change the canvas' header and/or footer. Header and footer definitions are associated with the last call to `StartPage()`, so starting a new page gives you an opportunity to call the `SetHeader()` and `SetFooter()` methods again with new values.

- Use the `canvas.Delete()` method to release the resources allocated to the canvas.

```
pReport.Delete()
```

How You Use the Artists

The methods in the derived Artist classes are used to obtain and set configuration information for the specified device, and then to render to that device. The methods in the Artist classes generate the report based on the information specified in an instance of Canvas.

To use an artist to generate a report:

1. Create a dynamic instance of the appropriate artist class.
2. Specify details about the particular output device (file, printer, or window) to use.
3. Render a defined canvas to the output device.

Create the Artist Instance

Call the `Create()` method for the derived Artist class that is appropriate for the type output that you want: `FileArtist`, `HTMLArtist`, `WindowArtist`, or `PrinterArtist`. It is useful to set a pointer to the value that the `Create()` method returns

Example:

```
pWArtist = WindowArtist.Create( )  
pFArtist = FileArtist.Create( )
```

Specify the Output Device

Each type of artist requires somewhat different configuration information to identify the specific device to which to output a canvas. You specify the output device by calling the appropriate method in the artist instance.

For FileArtist and HTMLArtist

Use `artist.SetFileName(filename, append)` to specify a file (HTML or plain text) to render the canvas to.

- The `filename` argument is a string containing the name of a file. If the file does not already exist, CA Aion BRE creates it before writing to it.
- The `append` argument is a Boolean representing whether to append to or overwrite the specified file. The default value is FALSE, indicating that the file should be overwritten.

Example:

```
pHArtist.SetFileName("c:\Reports\SalesRpt.html")
pFArtist.SetFileName("SalesRpt.txt", TRUE)
```

For WindowArtist

Use `artist.SetWindowHandle(window)` to specify the EditWindow to render the canvas to. The window argument is a pointer to an instance of EditWindow.

Example:

```
pWArtist.SetWindowHandle(AppWindow.ReportTextWin)
```

The EditWindow that you specify must have the multi-line style.

For PrinterArtist

If the output device is a printer, there are two options for specifying which printer to use:

Use `artist.SetPrinter(printer)` to set the printer to render output to. The printer argument is a string containing a system printer name.

Example:

```
pPArtist.SetPrinter("Engineering1")
```

Use `artist.QueryPrinter()` to prompt the user to select a printer.

Example:

```
pPArtist.QueryPrinter( )
```

Render a Canvas

Each derived Artist class has a `Render(canvas)` method for outputting the canvas instance that the canvas input argument identifies to the specified output device.

- The `FileArtist.Render(canvas)` method renders canvas to the text or HTML file set by `artist.SetFileName()`.

Example:

```
pFArtist.SetFileName("SalesRpt.txt", TRUE)
pFArtist.Render(pReport)
```

```
pHArtist.SetFileName("c:\Reports\SalesRpt.html")
pHArtist.Render(pReport)
```

- The `WindowArtist.Render(canvas)` method renders canvas to the window set by `artist.SetWindowHandle()`.

Example:

```
pWArtist.SetWindowHandle(AppWindow.ReportTextWin)
pWArtist.Render(pReport)
```

- The `PrinterArtist.Render(canvas)` method renders canvas to the printer set by `artist.SetPrinter()` or `artist.QueryPrinter()`.

Example:

```
pPArtist.QueryPrinter( )
pPArtist.Render(pReport)
```

Sample Application

In the Aion BRE \examples directory, the Doctor application illustrates Aion reporting functionality. This small application allows you to view a report about a patient in several different modes, and to send the report to a printer.

The same canvas instance can be output to each of the four output types: a text file, an HTML file, a window in the user interface, and a printer. You can view, run, and debug the sample application to see how the reports in this application are created and generated.

In addition to the Main entry class, the Doctor application contains two classes:

- Patient-Contains attributes defining the patient
- AppWindow-Defines the main application window

The AppWindow class contains all the attributes and methods that define the logic for the application, including:

- pCan-Attribute holding a pointer to the current canvas instance.
- CreateCanvas()-Method defining the canvas, that is, the body of the report. The body of this method is included in The Sample Canvas.

Various event methods associated with buttons on the application window. Clicking a button causes the canvas to be rendered to one of the four possible output options, as described in Output Options.

The Sample Canvas

The CreateCanvas() method in the AppWindow class defines the canvas for the Doctor application by creating a dynamic instance of Canvas and then calling the Canvas methods introduced earlier in this chapter to define the canvas' layout and contents:

```
var can is &Canvas
var p is &Patient

// Initialize the dummy data

p = Patient.Create( )
p.Name = "Joe Phoenix"

p.Diagnosis = "This patient has a dislocated Left Pinkie-Toe. Injury was
sustained during horse-play with patients pet crocodile. Please follow
instructions for proper treatment. Prognosis death within 6 months."

p.Instructions = List("Buddy tape",
"Ice for 15 Minutes",
"Stay off feet",
"Follow up with Podiatrist, Pediatrician, Oncologist, Veterinarian, and
Paleontologist in 1 week")

// Create the canvas

can = Canvas.Create( )
can.StartPage( )
can.SetName("County Medical Center")
can.SetHeader(FMT_DATE, FMT_NAME)
can.SetBackgroundColor("DarkCyan")
can.SetBackgroundImage("GrayBack.jpg")
```

```
// Write text and images to the canvas

can.WriteImage("CompanyLogo.jpg", "http://www.MyCompany.com")

can.FontBold( )
can.WriteCentered("Medical Report for " & p.Name)

can.FontNormal( )
can.WriteLine( )
can.SkipLine(2)
can.Write("Diagnosis: " & p.Diagnosis)
can.SkipLine( )

can.Write("Treatment Instructions: ")
can.WriteLineList(p.Instructions, FMT_NUMBERED)
can.SkipLine( )

// Write a table to the canvas

can.Write("Referrals:")

can.StartTable(List("Doctors Name", "Specialty"), List(20,20))
can.WriteLine(List("Dr. Scholl", "Podiatrist"))
can.WriteLine(List("Lolly Pop", "Pediatrician"))
can.WriteLine(List("Mel Anoma", "Oncologist"))
can.WriteLine(List("K. Nine", "Veterinarian"))
can.WriteLine(List("Dwight Bones", "Paleontologist"))

can.EndTable( )

can.SetFooter(NULL, "Page " & FMT_PAGE)
can.EndPage( )
p.Delete( )

return can
```

Output Options

By clicking a button on the Doctor application's main window, you can choose to render the canvas representing the patient's report to a window, a text file, an HTML file, or a printer. These choices are not mutually exclusive and the same canvas can be output in different ways in a single application.

Output the Report to a Window

Click the Report button to output the canvas to the application window's EditWindow:



The WhenbtnReportChosen event method contains the logic for rendering output to the EditWindow:

```

IF pCan = NULL
THEN
  pCan = CreateCanvas( )
END

VAR wArt is &WindowArtist

pApp.EditTest.setFont("Comic Sans MS")
pApp.EditTest.setText("")

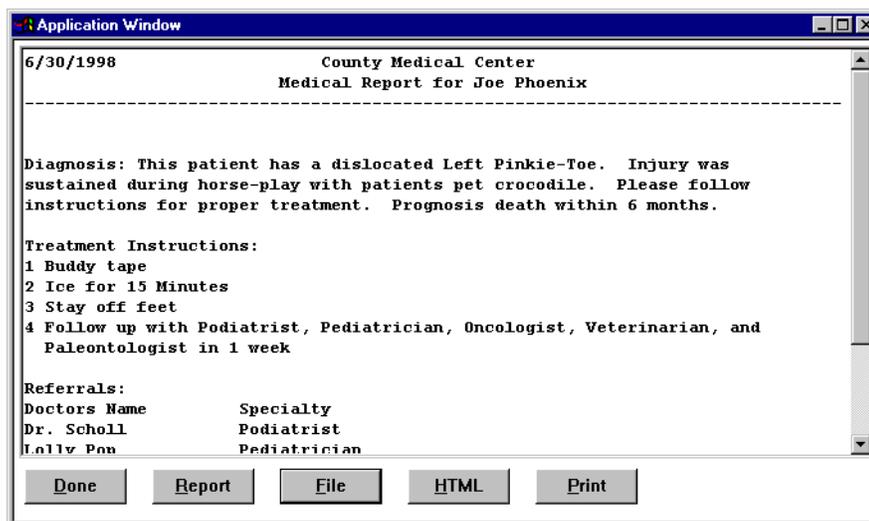
wArt = WindowArtist.Create( )
wArt.SetWindowHandle(pApp.editTest)
wArt.Render(pCan)

wArt.Delete( )

```

Output the Report to a Text File

Click the File button to output the canvas to a text file called Doctor.txt and to display the resulting text file in the EditWindow. The resulting file looks like this when displayed in the application window:



The WhenbtnFileChosen event method contains the logic for rendering output to a text file:

```

IF pCan = NULL
THEN
  pCan = CreateCanvas( )
END

VAR fArt IS &FileArtist

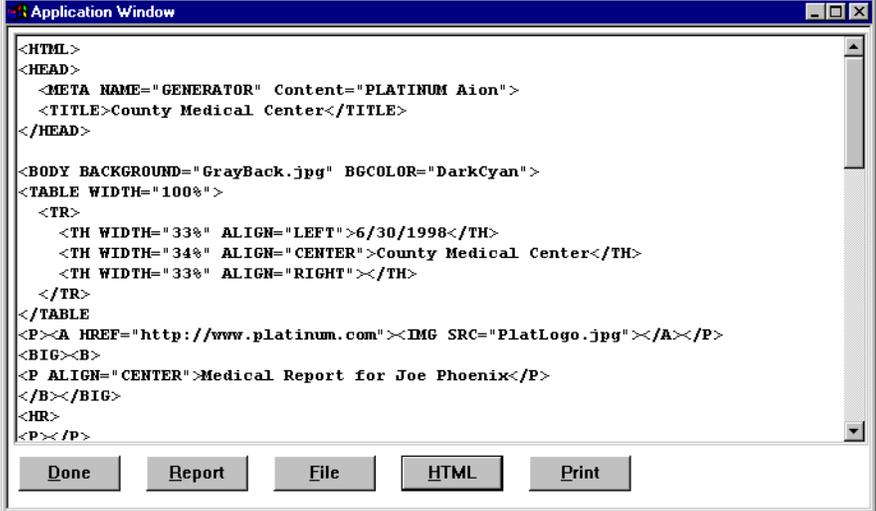
fArt = FileArtist.Create( )
fArt.SetFileName("Doctor.txt")
fArt.Render(pCan)
fArt.Delete( )

// Display Doctor.txt in the EditWindow

editTest.setFont("Courier New")
editTest.readFile("Doctor.txt")
  
```

Output the Report to an HTML File

Click the HTML button to output the canvas to an HTML file called Doctor.html and to display the resulting text file in the EditWindow. The resulting file looks like this when displayed in the application window:



```

<HTML>
<HEAD>
  <META NAME="GENERATOR" Content="PLATINUM Aion">
  <TITLE>County Medical Center</TITLE>
</HEAD>
<BODY BACKGROUND="GrayBack.jpg" BGCOLOR="DarkCyan">
<TABLE WIDTH="100%">
  <TR>
    <TH WIDTH="33%" ALIGN="LEFT">6/30/1998</TH>
    <TH WIDTH="34%" ALIGN="CENTER">County Medical Center</TH>
    <TH WIDTH="33%" ALIGN="RIGHT"></TH>
  </TR>
</TABLE>
<P><A HREF="http://www.platinum.com"><IMG SRC="PlatLogo.jpg"></A></P>
<BIG><B>
<P ALIGN="CENTER">Medical Report for Joe Phoenix</P>
</B></BIG>
<HR>
<P></P>

```

The WhenbtnHTMLChosen event method contains the logic for rendering output to an HTML file:

```

IF pCan = NULL
THEN
  pCan = CreateCanvas( )
END

VAR hArt IS &HTMLArtist

hArt = HTMLArtist.Create( )
hArt.setFileName("Doctor.html")
hArt.render(pCan)
hArt.Delete( )

// Display Doctor.html in the EditWindow

pApp.editTest.setFont("Courier New")
pApp.editTest.readFile("Doctor.html")

```

Send the Report Output to a Printer

Click the Print button to output the canvas to a printer. When the method executes, print status messages display. The user is prompted to select one of the available system printers to print the report to.

The WhenbtnPrintChosen event method contains the logic for rendering output to a printer, prompting the user to specify the printer to use:

```
IF pCan = NULL
THEN
  pCan = CreateCanvas( )
END

VAR pArt IS &PrinterArtist

pApp.editTest.SetText("Initializing...")

pArt = PrinterArtist.Create( )
pArt.QueryPrinter( )

pApp.editTest.SetText("Printing...")

pArt.render(pCan)

pApp.editTest.SetText("Done.")

pArt.Delete( )
```


Chapter 13: Generate and Use C and C++ Components

This chapter discusses generating Aion BRE components that can invoke or be invoked by applications written in C or C++.

When you create an Aion BRE application, you have these options:

- To call a previously written C function from within the Aion BRE application.
- To create an Aion BRE component as a server that can be called by a C or C++ client.

Note: You can also generate and use a variety of other interfaces, including managed C++, Java, and COM. These are documented in the corresponding chapters of this guide.

This section contains the following topics:

[Build an Aion BRE Component with an Interface Layer](#) (see page 335)

[Invoke Aion BRE Methods from C/C++ Clients](#) (see page 336)

[Invoke C Functions from Aion BRE](#) (see page 339)

[Data Type Mappings](#) (see page 341)

Build an Aion BRE Component with an Interface Layer

To enable clients to invoke methods of an Aion BRE-generated component, it is necessary to specify which Aion BRE classes have their public methods exposed as an application programmer interface (API). The idea of exposing public methods of a class or classes as an API is the fundamental principle behind Aion BRE interface layers. The methods of a class can be exposed as a C, C++, Managed C++, Java, or COM API. To specify which Aion BRE classes are exposed, set the Export property for each desired class (using the Export check box on the Class Properties page). All public instance methods in those classes will be exposed when the application is built with the selected interface layer.

To create an API for an Aion BRE component, follow this list:

- Check the Export Class checkbox on the Properties tab of the Class Editor.
- Specify the instance methods that you wish to expose as public for an exported class on the Properties tab of the Method Editor.
- Select the desired Interface Layer on the Build Directives tab of the Libraries Property dialog. To open the Build Directive tab:
 1. From within the open Aion BRE application, highlight the Libraries node in the Project Workspace tree.
 2. Right-click and choose Properties.
The Library Properties dialog displays.
This dialog is used to (optionally) specify application-specific information, such as build details and comments.
 3. Click the Build Directives tab. In the Interface Layer field, choose C or C++ from the drop-down list.

Note: All fields on this tab are optional. The remaining fields are discussed in the “Running and Building Applications” chapter.

You can build the application as usual by choosing the Build item from the File menu.

More Information

[Invoke Aion BRE Methods from C/C++ Clients](#) (see page 336)

Invoke Aion BRE Methods from C/C++ Clients

To enable a C or C++ client to invoke methods of an Aion BRE (server) component, select the C or C++ Interface Layer of the Build Directives tab of the Library Properties Dialog.

Build the Aion BRE server by choose the Build item from the File menu. The compiled Aion BRE application (EXE or DLL) exposes all public methods of exported classes so that a C or C++ application can access them.

When you generate an Aion BRE application with a C or C++ interface layer, a header (.h) file is produced. Include this .h file in your C or C++ application to be able to use the Aion BRE methods that have been exposed. Once you include the generated header file in your C or C++ program, you can access any of the Aion BRE methods it references. Because the information contained in the generated header file differs depending on whether you chose C or C++ for the Interface Layer, the convention for using exported Aion BRE methods differs depending on whether you are calling them from a C or C++ program.

More Information:

[Build an Aion BRE Component with an Interface Layer](#) (see page 335)

Use Exported Aion BRE Methods in a C Program

To invoke an Aion BRE component from a C program, include the generated *appname.h* file (where *appname* is the name of the Aion BRE application), and link with the generated *appname.lib* file. This section describes how to utilize the methods defined in the *appname.h* file to invoke those methods in the Aion server component.

Note: For special procedures for writing C components that access mainframe Aion BRE components, see the chapter “Build and Manage Aion BRE components” in the *CA Aion BRE Mainframe User Guide*.

Be sure to review the Data Type Mappings section to understand how data types are transferred between Aion and the invoking C client. The discussion includes special consideration on handling strings.

Procedure Overview

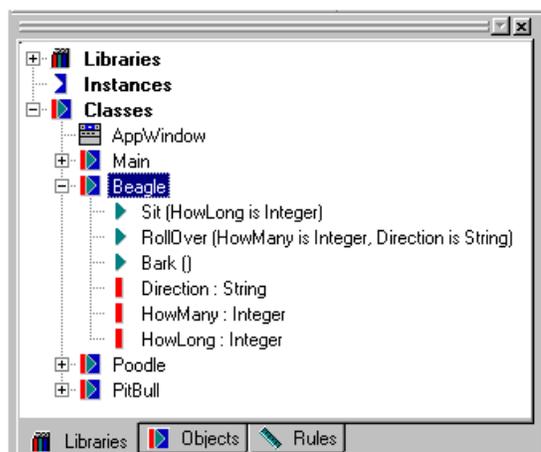
For C, follow these general steps to use exported Aion BRE methods.

- Create a class instance (`val_inPtr`) by a passing the name of the instance to class's `Create(name)` method.
- Invoke the Aion BRE methods, passing in the pointer to the class instance.
- When finished, delete the class instance using `class_delete()`.

At compile-time, include the `.H` file and link with the `.LIB` file.

Example:

This section demonstrates sample C code for accessing an Aion BRE application called *DogSchool*, which has been compiled with a C interface layer, and which contains the following objects:



Example:

The following statements demonstrate C code that can be written to access the Aion BRE methods:

```

/* pass in the pointer to create a class instance */
val_inPtr pBeagle = Beagle_Create("Snoopy");
/* pass the class-instance-pointer to the methods */
Beagle Sit (pBeagle);
Beagle Stay (pBeagle, howLong);
Beagle RollOver (pBeagle, howmany, direction);
.
.
/* delete the class instance */
Beagle Delete (pBeagle)

```

Use Exported Aion BRE Methods in a C++ Program

CA Aion BRE supports class usage in the same way as the C++ language does. Because of this, Aion BRE classes can be accessed in C++ in the same way that native C++ files are used.

To use exported Aion BRE methods in a C++ program

1. Include the Aion *appname.h* header file with your C++ application.
2. Instantiate exposed Aion BRE objects using `new classnm()` (where *classnm* is the name of the exported Aion BRE class). The handle is returned as a pointer *pClass*.

Aion BRE classes can be used in the same way as traditional C++ classes. Use `new` and `delete` functions to create and delete objects.

Example:

```
Beagle *pBeagle = new Beagle( );  
pBeagle -> Sit( );  
delete pBeagle;
```

Review the Data Type Mappings section to understand how data types are transferred between Aion and the invoking C++ client. The discussion includes special consideration on handling strings.

Invoke C Functions from Aion BRE

From an Aion BRE application, you can also invoke functions written in C. To invoke an external C function from Aion, you must know the name and location of the C library file, as well as the name and calling convention for the referenced C procedure.

Many methods defined in the Aion BRE-supplied libraries happen to be external methods. They are used to access low-level system functions to control the client environment. For example, `WinLib` serves as an interface to the graphical user interface functions of Windows, while `DataLib` provides access to the interface functions of various database drivers.

Before accessing an external C method, it must have already been implemented in C and linked to a DLL.

Note: Both class and instance methods can be external methods. There are no specialization constraints. You can specialize a method to be either internal or external.

How to Create an External Method in Aion BRE

Once the DLL containing the external C implementation is created, use the Aion BRE Method Editor to define the new Aion BRE method. Unlike an internal method, you must specify the following information on the Method Properties page:

- Check the External Style for the method. This enables the External Method Definition section on the lower right side of the Properties dialog.
- Provide the External Method Definition (Convention, Procedure Name, and Library Name)
 - To create an API based the standard calling convention for function arguments, select Standard Call in the Convention drop-down list.
 - To create an API based on the C calling convention (`_cdecl`), select C in the Convention dropdown list. The C calling convention is the more commonly used convention for C functions.
 - In the Procedure Name field, type the exact name of the C function. For example, *PrintThis*.

Note: Because the name of the function within the DLL is supplied in the method's Procedure Name property, you can use any name for the Aion BRE method itself.

Arguments specified as input or output arguments must agree with the arguments specified in the external implementation in order, data type, and size. The return value must agree in data type and size with the return value specified in the external implementation. Use Binary for complex types.

- In the Library Name field, enter the exact name of the library file housing the C function. This file is usually a DLL. Specifying the filename extension is optional.

To ensure that arguments are specified properly, you can view the prototype for the function. Click Prototype on the External Method Definition group box. The typical external function prototype for the C calling conventions is:

```
<return-datatype> _cdecl <proc-name> (<arglist>);
```

You have now created an Aion BRE method that takes its logic from a pre-existing C function. At runtime, Aion dynamically loads the function from the specified .DLL file.

Call an External Method (Runtime)

The syntax for calling an external method is the same as that for calling an internal Aion BRE method.

When an external method is called at runtime, CA Aion BRE transfers control to the external function, passing along the input and output arguments. If the Aion BRE application is dynamically linked, the Aion runtime environment automatically follows these steps to transfer control to the external function:

To call an external method:

1. On first use of the DLL containing the C function, the DLL is loaded dynamically using LoadLibrary. The DLL location must be specified on the system library path.
2. Using the module handle for the DLL, the entry point for the function is resolved by name using GetProcAddress.
3. Values for input arguments and addresses for output arguments are pushed onto the program stack. The stack is built using the specified calling conventions for the external method.
4. Control is passed to the function.

The return value and the values of any output arguments are transferred back to the application after the function completes execution.

Review the Data Type Mappings section to understand how data types are transferred between Aion and C functions. The discussion includes special consideration on handling strings.

Note: These actions do not apply for applications that are statically linked or running interpreted.

Data Type Mappings

When writing an external method or calling a class or instance method, strictly observe argument conventions. Improperly specified arguments lead to fatal errors during execution. Because calls to and from an Aion BRE application are dynamic, no type checking can be performed to verify the correctness of the argument list or of the return value. Be very careful when defining external methods, or when using a 3GL to interact with the running application.

Input Arguments

Input arguments are passed by value. For example, the Aion integer is a 4-byte quantity. When passing an input integer argument to a method, use LONG. Preferably, use the definitions in *appname.h* to declare your variables. Constant values can be passed as input arguments (for example, NULL). Be sure to cast constant values to the appropriate data type when calling back to the running application.

More Information:

[Strings as Input Arguments](#) (see page 345)

Output Arguments

All output arguments are passed by reference (the address of the value).

More Information:

[Strings as Output Arguments](#) (see page 346)

Return Values

Return values are passed by value. However, additional work must be done when returning string values.

More Information:

[Strings as Return Values](#) (see page 348)

Mapping Between Aion BRE and C Data Types

The following table shows the mapping between Aion BRE data types and C data types when defining arguments and return values for external methods and when passing arguments to internal methods:

Aion BRE element	C data type
string	LPSTR
out string(n) *	LPSTR, UINT
integer	LONG
integer(2) *	short

Aion BRE element	C data type
integer(1) *	signed char
real	double
real(4)	float
Boolean	BOOL (unsigned int)
date	typedef structure {short yr; WORD mon, WORD day}
time	LONG (time_t)
list of <data type>	void *
pointer to <class>	void *

*valid only for external method argument definitions

Aion BRE Strings in C and C++

CA Aion BRE uses a different internal string format from C/C++. The internal Aion BRE string format is defined by `val_Str` or `str_Ptr`, which are defined in `opsys.h`. Unless the Aion BRE string is converted to the format of a C string, the program will see it as different string. The `opsys.h` file also contains definitions for the string manipulation functions required to convert an Aion BRE string to and from a C string as well as other useful string manipulation functions. These functions begin with the `xs_`. To use these functions, the C/C++ program must be linked with the current version of `resysRN.lib`, where *RN* is the latest Aion BRE release number.

The following table summarizes these functions:

Function	C Return Type	Arguments	Purpose
<code>xs_Add</code>	VOID	<code>str_Ptr *str,</code> <code>LPTSTR zs</code>	Add a C or C++ string <code>zs</code> to the end of an Aion BRE string <code>str</code> .
<code>xs_Allocate</code>	<code>val_Str</code>	<code>val_Int imaxlen</code>	Create and return an Aion BRE string of maximum length and actual length equal to <code>imaxlen</code> .
<code>xs_Assign</code>	VOID	<code>LPTSTR zdest,</code> <code>UINT maxlen,</code> <code>LPTSTR zs</code>	Copy the source string, <code>zs</code> , to destination, <code>zdest</code> up to maximum length of <code>maxlen</code> .
<code>xs_Compare</code>	INT	<code>LPTSTR z1,</code> <code>LPTSTR z2</code>	Compare <code>z1</code> and <code>z2</code> for equality. (Case insensitive) Returns 1 if <code>z1</code> is greater than

Function	C Return Type	Arguments	Purpose
			z2, -1 if z1 is smaller than z2, 0 if both strings are equal; and -2 if one string is NULL and the other is not.
xs_Concat	str_Ptr	LPTSTR z1, LPTSTR z2	Create and return an Aion BRE string by concatenating strings z1 and z2.
xs_Dispose	VOID	str_Ptr *str	Dispose of a string.
xs_FixLength	BOOL	str_Ptr str	Find the Aion BRE string's C-style zero-termination length. Re-define Aion BRE string length based on this C length. Returns True if successful; false otherwise.
xs_Index	LONG	LPTSTR zbase, LPTSTR ztest, LONG lstart, BOOL reverse	Return the position of string ztest as substring within zbase, starting at position lstart. Search from the end if reverse is true.
xs_ItoS	val_Str	val_Int ival	Create and return an Aion BRE string of length 1 and the first character having ascii value equals to ival.
xs_Length	LONG	LPTSTR zs	Return the length of zs.
xs_Make	str_Ptr	LPTSTR zs	Create and return a new Aion BRE string with the value of a C or C++ string zs.
xs_New	VOID	str_Ptr *str, UINT len	Pre-allocate specified length of storage for a new, empty Aion BRE string
xs_NextWord	val_Str	LPSTR zsrc, LPTSTR zdel, LONG *lpos	Create and return an Aion BRE string from a substring extracted from zsrc, starting at position pointed at by lpos, and include up to the end of the substring zdel in zsrc after the start position. Update lpos to point to a position in zsrc after the end of substring zdel. Trim spaces from the Aion BRE

Function	C Return Type	Arguments	Purpose
			string.
xs_Overwrite	VOID	LPTSTR zdest, LPTSTR zs, LONG lidx	Replace a portion of a string, zdest, with string zs beginning in position lidx.
xs_PtoZ	LPTSTR	str_Ptr str	Convert an Aion BRE string to a C or C++ string. Return the C or C++ string.
xs_Replace	VOID	str_Ptr *dest, LPTSTR za	Replace content of Aion BRE string dest with content of C or C++ string zs.
xs_SetMaxLength	VOID	str_Ptr *str, UINT len	Set the maximum length for the storage of an Aion BRE string. Length of the actual content of Aion BRE string is not changed.
xs_StoI	val_Int	LPTSTR zs	Return the ascii value of the first character of string zs.
xs_Substring	str_Ptr	LPTSTR zs, LONG start, LONG cnt	Create and return an Aion BRE string by extracting a substring of length cnt from string zs, starting at position start.

Strings as Input Arguments

Unlike other data types, strings are passed by reference, not by value, through the type LPSTR.

This means that if CA Aion BRE calls an external C function, the C function could potentially change the input string. This is not recommended as the called function does not know the size of the allocated string. Strings that will be changed by the caller should be defined as Output arguments in Aion BRE. However, there are times when you may need to call C functions whose signature you cannot change.

Example:

The Windows API call, `GetUserNameA`, has a function in `advapi32.dll` with a prototype as follows:

```
VOID __stdcall GetUserNameA (LPSTR s,LPDWORD s_max);
```

Where `s` references the string in CA Aion BRE which will hold the output, filled in by the DLL, and `s_max` specifies the length of the string filled in by the DLL.

To generate the appropriate Aion BRE prototype (using the Standard Calling convention), you must declare the string as an INPUT string and "preallocate" the string by padding it to a sufficient width before the call. For the length argument of type `LPDWORD`, use an Aion output integer.

Create an external method in Aion:

```
GetUserName(IN s string, OUT maxlen integer)
```

(with Procedure Name = "GetUserNameA" and Library Name = "advapi32.dll").

Invoke the method as follows:

```
var s string
var maxlen integer
//"pre-allocate" space for the "output" string
s = format(" ",NULL,NULL,100)
GetUserName(s,maxlen)
```

The string `s` will now contain the user name and `maxlen` will contain the length of `s` that is passed back from the external function call.

Strings as Output Arguments

String output arguments are handled differently depending upon whether Aion BRE is calling C or being called from C/C++.

For the scenario where Aion BRE is calling C, the address to the character buffer (`LPSTR`) is passed in the same way as an input string argument. However, additionally the maximum size of the string is passed to the external program as an argument of type unsigned int.

Example:

You might have a C function whose prototype is as follows:

```
VOID _cdecl getMessage (LPTSTR s, UINT smax);
```

where *s* is the output string, filled in by the DLL and returned to Aion in the character buffer provided by Aion, and *smax* specifies the maximum length of the character buffer (including the NULL termination character).

To generate the appropriate Aion BRE prototype for this function, create a method:

```
getMessage(OUT s string)
```

Invoke the method as follows:

```
var s string  
getMessage(s)
```

Note: that the string length is not explicitly specified in the Aion BRE method call, it is added automatically for output strings and passed as an additional argument when Aion BRE calls the external function.

When a string output argument is specified without a size, it has a size of 1024. However, this default size may be overridden by specifying a size in parentheses as part of the argument.

For example, if `getMessage` were defined this way on the Aion BRE side: `getMessage(OUT s string(30))`, the size of the string buffer would be set to 30 and the value of `smax` passed into the C function would be 30.

For the scenario in which Aion is being called from C or C++, an output string needs to be defined on the C side as a pointer to a `val_Str`. The C program creates a `val_Str` and passes the pointer to it to the Aion program, which the Aion program then fills in with the output string.

Assume that you have an Aion BRE method defined such as: `getMessage(OUT o string)`

Example:

This is an example of calling it as an Aion BRE method from C:

```
val_Str v1;
LPSTR s1;
xs_New(&v1,100);
aionStuff->getMessage(&v1);
s1 = xs_PtoZ(v1)
```

To compile and link the C program, you must include `opsys.h` and link with `resysRN.lib`.

Strings as Return Values

Aion BRE strings are represented in C/C++ using the `val_Str` construct. When returning a string to Aion from a called C/C++ function or when returning a string to C from a called Aion BRE method, you must use one or more of the `xs_` functions to do the conversion to a standard C string:

Consider this simple C function called as an external method from Aion BRE.

```
DllExport val_Str getValue ()
```

The C code should follow the following pattern.

Note: that in the C code, we require the extern "C" statement; otherwise the C++ compiler will mangle the function prototype.

```
#include "opsys.h"
#include <stdio.h>
#define DllExport    __declspec( dllexport )
#ifdef __cplusplus
extern "C" {
#endif
DllExport val_Str getValue ()
{
    char s[100];
    sprintf(s, "We are returning a value");
    return xs_Make(s);
}
#ifdef __cplusplus
}
#endif
```

In this case you use xs_Make() to create an Aion BRE string from a C string. Similarly, if you want to return a C string from a called Aion BRE method, you must use xs_PtoZ() to do the conversion as in this snippet of C code:

```
LPSTR s;
val_Str valStr = hello->getValue();
s = xs_PtoZ (valStr); // convert to null-terminated string
```

To compile and link this second example, you must include opsys.h and link with resysRN.lib.

NULL Values

The opsys.h header file contains the definitions that map the internal Aion BRE representations for NULL values of different data types as follows:

Data type	NULL Constant
Boolean	NULLBOOL
integer	NULLINT
real	NULLREAL
All other types	NULL

Note: Booleans are a special case for Aion BRE. In C, a Boolean is FALSE if it equals zero, and TRUE if it contains anything else. In Aion BRE, Booleans are assigned differently. A Boolean is FALSE if it equals 0, TRUE if it equals 1. Aion Booleans also have null values (NULLBOOL) defined as 2. If you are using external C or C++ components with Aion BRE applications, keep in mind that a C Boolean equal to 3, for example, is not understood by CA Aion BRE.

Chapter 14: Generate and Use Managed C++ Components

CA Aion BRE provides a Managed C++ interface layer as an additional service that opens the door to the world of .NET.

The Managed C++ interface enables CA Aion BRE components to be called from client applications written in any .NET compatible language (for example, Managed C++, C#, and Visual Basic.NET.)

This section contains the following topics:

[The Managed C++ Interface Layer](#) (see page 351)

[Create an Aion BRE Component with the Managed C++ Interface](#) (see page 355)

[Application Programming for .NET: The Basics](#) (see page 357)

The Managed C++ Interface Layer

The Managed C++ interface layer allows programmers to deploy CA Aion BRE components within the .NET environment. This section describes the basic structure of the Managed C++ interface layer.

Managed Code

Managed code derives its name from its relationship to the .NET runtime environment, called the Common Language Runtime (CLR). The CLR is functionally comparable to the Java Virtual Machine (JVM) runtime environment. However, unlike the JVM, which supports only the Java language but covers multiple platforms, the CLR supports a wide range of languages that implement a published common language standard but is currently available only on platforms running the Windows operating system. Microsoft or third party vendors may provide CLR's for other platforms in the future. The CLR provides management facilities for applications running within this environment. These facilities include such capabilities as automatic garbage collection, type safety guarantees, security checking, and execution and just-in-time compilation. CLR-managed languages must be capable of being represented in Microsoft's Common Intermediate Language (CIL), which is the language that the CLR compiles when executing applications written in managed code.

From the .NET perspective, there are two types of programming languages: *managed* and *unmanaged*. All traditional languages, including traditional C++, are unmanaged, because they cannot execute under the management facilities of the CLR. Languages that produce managed code include Microsoft's own C#, a very Java-like language that's been given a C-like name, and Visual Basic.NET, a significant recasting of the popular Visual Basic language. There are also versions of such common languages as COBOL, Eiffel, Fortran, RPG, Smalltalk, and several version of Pascal as well as a number of less standard languages (Oberon, Perl, Python) that are .NET compatible.

Important! It is possible for managed code to incorporate unmanaged code. The unmanaged code must take full responsibility for managing its own resources (for example, performing its own garbage collection). The Managed C++ interface layer in CA Aion BRE makes use of the ability of managed code to call unmanaged code, as will be shown in the section The Structure of Managed C++ Interface Layer.

What is Managed C++?

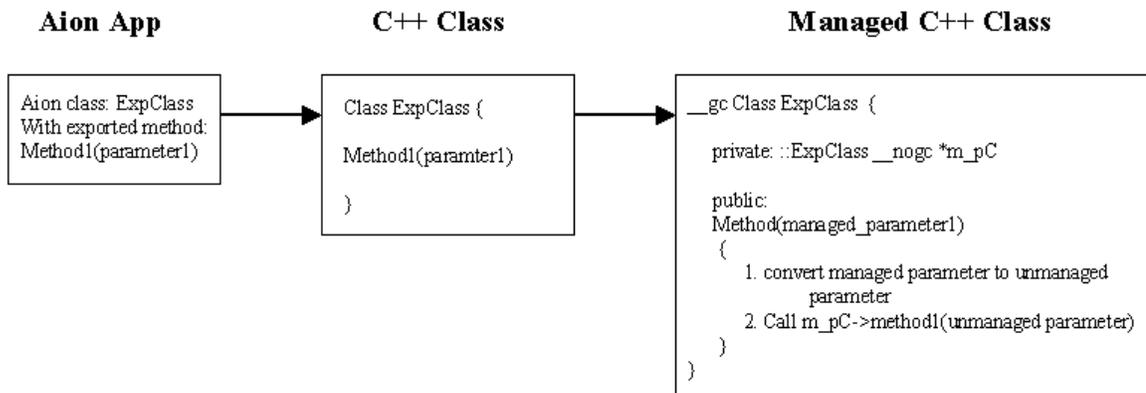
Managed C++ is the Microsoft C++ language with newly added keywords and features (called **managed extensions**) to support .NET programming. Managed C++ is essentially C++ code that takes advantage of all the CLR features. New keywords included within the Managed C++ language include **__gc**, which designates that a class should be subject to the automatic garbage collection facility of the CLR, and **__nogc**, which excludes a portion of the Managed C++ code from the automatic garbage collector. Thus it is possible to mix managed and unmanaged code in C++ programs.

Given the elegance of the C# language, it is doubtful that a programmer would choose Managed C++ as the first choice for a programming language. Fortunately, CA Aion BRE generates all necessary Managed C++ code to deploy a CA Aion BRE component as a DLL that will be executable within the CLR. Once deployed under the CLR, any .NET language will be able to access that component.

Structure of the Managed C++ Interface Layer

The Managed C++ interface layer of CA Aion BRE rests upon the C++ interface layer. However, this layer is completely hidden; it is wrapped by a layer of generated Managed C++ code.

The structure of the Managed C++ interface layer is illustrated in the following diagram.



Each exported CA Aion BRE class has a C++ counterpart. This C++ class is embedded in a garbage-collected class in the Managed C++ code, and each method is wrapped by a Managed C++ method. The Managed C++ interface generates a .NET proxy object for each exported class in the CA Aion BRE application. Notice that the code generation also includes data type conversion from:

- CA Aion BRE types to C++ (and the reverse)
- C++ types to .NET (and the reverse)

More Information:

[Generate and Use C and C++ Components](#) (see page 335)

[Data Conversions and Exception Handling](#) (see page 371)

Generate the Managed C++ Interface Layer

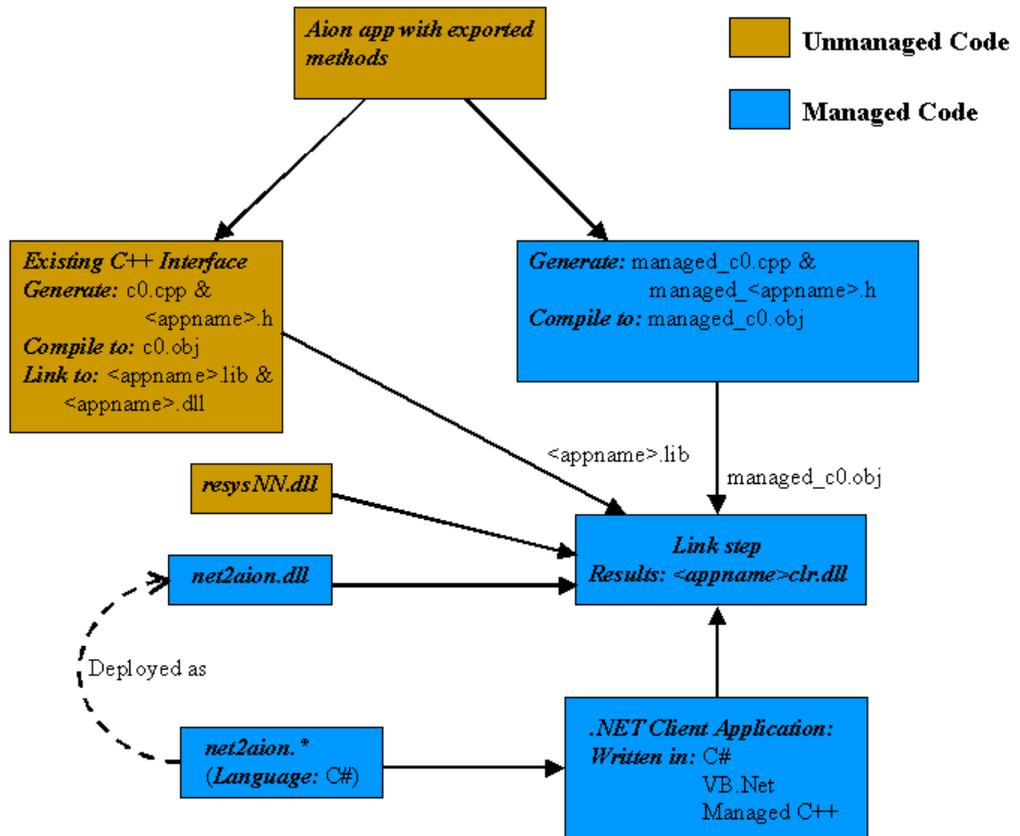
The major processes of Managed C++ interface generation are

- Code generation for C++ interface
- Code generation for Managed C++
- Compiling and linking the C++ interface code
- Compiling the Managed C++ interface code and linking it with the C++ dll and other required software.

Besides the C++ and Managed C++ code that is generated for each CA Aion BRE application, the Managed C++ interface layer also requires:

- ResysNN.dll (where "NN" represents the current version of CA Aion BRE).
- The Net2Aion project. The Net2Aion provides base classes to implement session safety in the .NET environment. All managed classes will derive from AionObject as defined in this project.

The generation process is illustrated in the following diagram.



The managed code, `managed_c0.cpp` and `managed_<appname>.h`, is compiled with the Common Language Runtime and it generates the `<appname>.dll`. The generated dll is a valid .NET assembly. Any .NET language can now include this assembly and use the classes exported from Aion BRE. It is necessary to include the `net2aion` project, provided by CA Aion BRE, within the .NET client application.

More Information:

[Application Programming for .NET: The Basics](#) (see page 357)

Set up the Environment

Visual Studio .NET for C++ and the Microsoft .NET Framework 2.0 should be installed.

Create an Aion BRE Component with the Managed C++ Interface

In this section we will create a simple Aion BRE application that is called from a Managed C++ client (see the `HelloDotNetWorld.app` file in the Managed C++ examples that are provided with Aion).

Code an Exported Class

We construct the classes whose public methods will define the interface of the CA Aion BRE component.

Example of HelloDotNetWorld elements:

- A class, for example, `AionWorld`.
- An attribute, `f_message`. In the example, this attribute has an initial value of "Hello DotNet world."
- Public get and set accessor methods.

Note: To insure correct construction of the accessor methods it is advisable to use the Create Accessor wizard from the Create Accessor option on the Logic menu.

Export the class by checking Export on the Properties tab of the Class editor.

Note: The Export Properties button is not relevant for the Managed C++ interface layer.

Build the Aion BRE Application

To build the Aion BRE application, select the Managed C++ interface layer from the Build Directives tab of the Properties dialog of the library. Insure that the Executable directory on the Directories tab of the Library properties is set to “..\exe”.

CA Aion BRE automatically copies all the necessary dlls to the executable folder specified on Directories tab. These files are:

File	Source	Function
HelloDotNetWorld.dll (AppName.dll)	Generated	The compiled Aion BRE application; deployable under the CLR
Net2Aion.*; Net2aion.dll	Provided	DLL containing common behavior for Managed C++ Aion BRE objects
Log4net.*	Provided	.NET package for debugging a .NET client application
resysnn.dll	Provided	Aion BRE runtime dlls.

When deploying the Managed C++ server application, the location of these dlls must be in the directory of the server application or in the systems PATH.

Write and Compile the .NET Client

Now create .NET client application, for example, “MyMPCCTest.exe” in the executable directory.

Note: The client may be written in any .NET language.

The .NET client may consist of a single class that needs to perform primarily functions for:

- Creating an instance of the exported Aion BRE application class (AionWorld).
- Calling an exported method of the class.

In addition to using the appropriate .NET namespaces for the application (for example System.Windows.Forms), your application must also use the Net2Aion project and the project for your Aion BRE application (which is generated by the Managed C++ interface layer).

Note: Although the Net2Aion project is coded in C# and the project of your Aion BRE application is generated in Managed C++, the powerful feature of the .NET environment is that it allows the use of such language integration for .NET compatible languages.

Example:

The following code segment presents a simple example in which C# code invokes the exposed method of an Aion Managed C++ server:

```
using Net2Aion;
using HelloDotNetWorld;
...
private void invokeAion() {
    //Aion invocation is here:
    AionSession aionSession = AionSession.GetInstance();
    AionWorld aionWorld = new AionWorld(aionSession);
    aionWorld. setMessage("textBox1.Text");
    listBox1.Items.Insert(0, aionWorld. getMessage( ) );
}
```

(Your actual code may differ from this depending on what you are doing in your user interface, if any.)

Compile the client application with the appropriate compiler for the selected .NET language.

More Information:

[Application Programming for .NET: The Basics](#) (see page 357)

Deploy a Managed C++ Component

Aion BRE-generated Managed C++ components cannot be deployed in the Global Assembly Cache (GAC).

Application Programming for .NET: The Basics

The essential lesson to know about calling a CA Aion BRE server from a .NET client is that the client must obtain a session object of type `AionSession` in order to bind the CA Aion BRE objects. Every object has to be created with a reference to an `AionSession` object. The `AionSession` object is the means by which CA Aion BRE can guarantee thread safety in a multi-threaded environment such as .NET provides.

`AionSession` is defined in the `Net2Aion` project through the parent `AionObject` class. Three constructors will be generated for each class that is exported from the CA Aion BRE application and exposed in the Managed C++ dll:

```
ExpClassA(Net2Aion::AionSession *session);
ExpClassA (String __gc *objectname, Net2Aion::AionSession *session);
ExpClassA (__int64 AionAppExp, Net2Aion::AionSession *session);
```

These constructors help in creating the managed object in the specified AionSession. These constructors in turn calls the protected member CreateAionObject() in the Managed C++ dll, which creates the private Aion BRE object (this object is represented by the m_pC pointer in the managed_c0.cpp code, see Structure of the Managed C++ Interface Layer.

Once an AionSession object is obtained, this object is used in the constructor of the proxy class (AionWorld).

```
myAionApplication. anAionWorldObject myObject = new  
    myAionApplication. anAionWorldObject(mySession);
```

In the preceding code, anAionWorldObject is an exported class from the Aion BRE application, which is replicated as a proxy class in myAionApplication.

Once an object is created in the .NET client, the exposed methods of the CA Aion BRE class can be called through that instance.

Example:

```
anAionWorldObject.setMessage("Hello Aion world");
```

where setMessage() is a public method of anAionWorldObject.

Note: References to non-exported classes in the .NET code causes compile errors.

More Information:

[Thread Management](#) (see page 382)

[Structure of the Managed C++ Interface Layer](#) (see page 353)

Support for Output Parameters

The Managed C++ interface layer supports output arguments for the following basic data types

- Integer
- Real
- Boolean
- String

An argument of type OUT in an Aion BRE method signifies that it is both an input and output argument. In the .NET environment such an argument can be mapped to reference parameters, defined with keyword **ref**. The **ref** method parameter keyword on a method parameter causes a method to refer to the same variable that was passed into the method. Any changes made to the parameter in the method will be reflected in that variable when control passes back to the calling method.

Example:

Consider an Aion BRE method TestIntegerOutputArg with arguments

```
out out_i is integer
in in_i is integer
```

Where *out_i* is an input and output argument and *in_i* is an input only argument. This method is implemented in the Managed C++ interface as:

```
void TestIntegerOutputArg (Int64 __gc *out_i, Int64 in_i);
```

The method TestIntegerOutputArg can be invoked from C# code as follows:

```
long output_value = 99;
long integer_input_value = 111;
anAionObject.TestIntegerOutputArg(ref output_value, integer_input_value);
```

Support for Complex Data Types

The Managed C++ interface layer supports passing complex types between the client and .NET components. Complex data types are types that are defined as consisting of a collective of other complex or primitive data types. For example, a Person may be a complex data type consisting of a number of primitive data types of strings and integers (for Name and Age respectively) and complex data types (such as Address, which may consist of separate strings for Street, City, State, and more). In traditional object-oriented programming, complex data types are supported by defining the type as a class. To support complex data types under the Managed C++ interface layer, construct the class defining the complex in the following manner:

- Define public methods of class that will serve as the exported class for the web services in terms of pointers to other classes that define your complex data types, for example, `processInfo(&InputData) : &Info`, where "InputData" is a complex data type (class) describing input to the `processInfo` method and "Info" is a complex data type (class) describing the information returned by the method.
- Insure that the classes defining complex data types are exported.
- Define public accessor methods on each class defining a complex data type. The accessor methods may involve either primitive data types or further complex data types (that is, pointers to further classes defining complex data types).

Object Management Under .NET

This section discusses object management from the perspective of writing the .NET client that accesses the Managed C++ component generated from a CA Aion BRE server.

For examples, when a client creates a .NET object, a C++ object is created in the background and that in turn creates an Aion BRE object.

CA Aion BRE programmers are accustomed to using a class's `delete()` method to clear instances in Aion after they are no longer needed. The CLR favors automatic garbage collection. When the .NET object is garbage collected the destructor of the Aion C++ wrapper object will be called.

There are situations in which it will be desirable for the .NET client program to delete instances of its proxy classes. For example,

- The instance of the proxy class is created and the Aion BRE object is created behind the scenes. Responsibility for object management in this case falls upon the .NET programmer. The .NET programmer must delete the proxy instance, which should delete its associated object in Managed C++. C++, and CA Aion BRE server.
- Aion BRE internally creates an instance of an exported class (for example, through class containment), and a bit latter the .NET application asks for all instances of that class. Aion BRE returns handles to the generated .NET instances and the proxy class is instantiated on the fly (lazy instantiation). The .NET environment can make inquiries of the newly created instances of the proxy class. The question is now, Who is responsible for the Aion BRE created instance? Aion BRE does not know that the outside world created a reference to the instance. The .NET application does not know if Aion BRE would like to keep the object. In this case, it is possible for .NET client to delete the instance without deleting the associated Aion BRE instance.

In general, the philosophy should be that the creator of the object has the responsibility of deleting it:

Object Creator	Responsibility for Object Management
Aion BRE creates object	Aion BRE deletes object
.NET creates object	.NET deletes object
Aion BRE proxy is created dynamically	.NET object can be deleted; however the associated Aion BRE object will not be deleted.

Note: The attempt to reference an instance of a proxy class for which the associated object in Aion BRE has been deleted will result in the .NET environment throwing an unhandled Exception with the following message:

System.NullReferenceException: Object reference not set to an instance of an object.

Data Type Conversion

The Managed C++ interface layer supports the following conversion from .NET data types to Aion BRE data types.

Aion BRE Data Type	.NET Data Type
Boolean	System.Boolean
Integer	System.Int64
Real	System.Double
String	System.String
(an instance of) <ExportedClass>*	(an instance of) <ExportedClass>
Date	Not Supported
Time	Not Supported
List of Boolean	System.Boolean[]
List of Integer	System.Int64[]
List of Real	System.Double[]
List of String	System.String[]
List of (instances of) <ExportedClass>	<ExportedClass>[] (instances in)

* Name of the exported class.

Chapter 15: Generate and Use Java Components

CA Aion BRE provides a Java interface layer as an additional service that opens the door to the world of Java programming (for example, XML and web-enabled Aion BRE rule-based processing). The Java interface layer can be selected from the Interface Layer drop-down box on the Build Directives tab of the Library Properties dialog. The Java interface layer permits an Aion BRE compiled knowledge base (DLL) to be called from Java.

The CA Aion BRE solution to web-enablement is web-server independent and follows mainstream standards.

More Information:

[Aion BRE Deployment on the Web](#) (see page 376)

This section contains the following topics:

[The Java Interface Layer](#) (see page 363)

[Create an Aion BRE component Using the Java Interface](#) (see page 366)

[The Basics of Java Application Programming](#) (see page 369)

[Aion BRE Deployment on the Web](#) (see page 376)

[Thread Management](#) (see page 382)

[Additional Information](#) (see page 385)

The Java Interface Layer

The Java interface layer allows Aion BRE programmers to export classes and their public methods for use by a Java application or servlet. This section describes the basic structure of the Java interface layer and the way in which a Java application must be programmed in order to call an Aion BRE component.

Note: CA Aion BRE currently supports only a Java-calling-Aion-BRE.

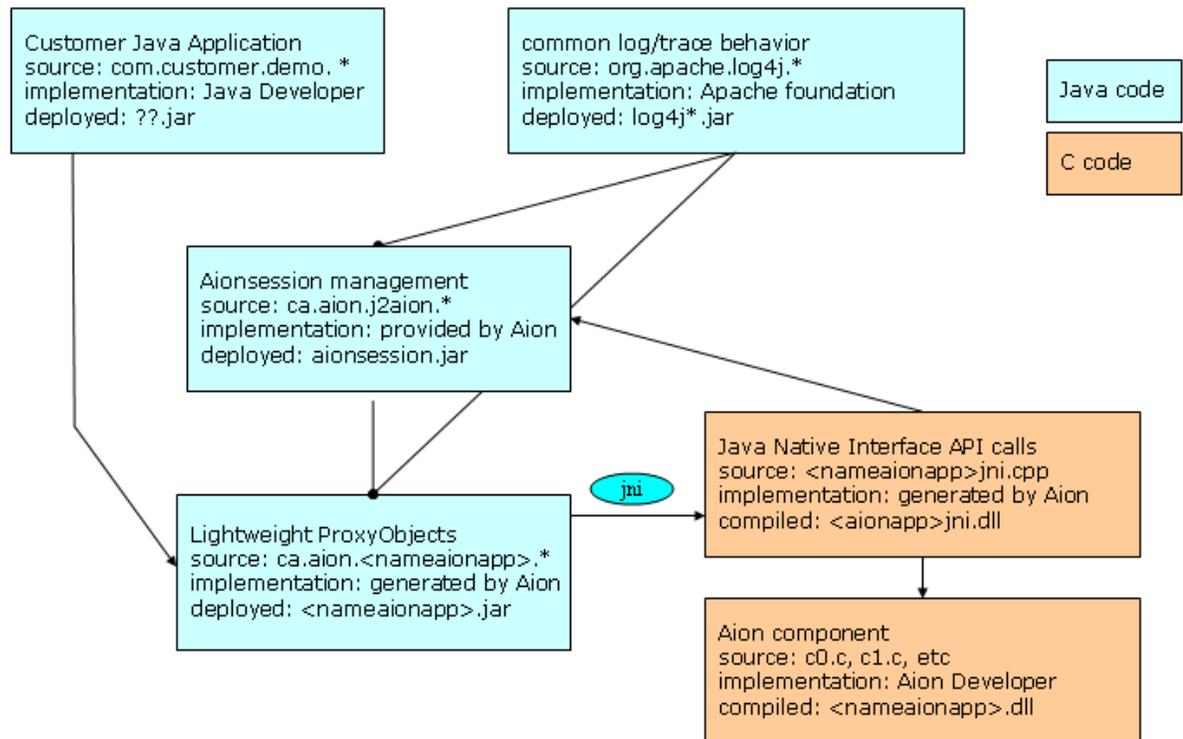
Elements in the Java Interface Layer

CA Aion BRE builds the Java interface layer from exported Aion BRE classes. During the build process, CA Aion BRE generates the Java code for the Java classes that correspond to the Aion BRE exported classes. These Java classes, which are derived from an Aion-provided Java class called AionObject, act as proxies for the Aion BRE exported classes. The Aion build process will also generate a C-based dll that maps the proxy objects in the Java code to the exported Aion BRE classes and converts data values between Java types and Aion BRE types. This C-based dll uses the Native Java Interface (JNI).

Thus, the principal elements in the Java interface layer are:

- The Aion BRE knowledge base component: <nameaionapp>.dll
- The C/JNI library: <nameaionapp>jni.dll.
- The Java definitions of the proxy classes: <nameaionapp>.jar

The following diagram depicts the structure of the Java interface layer. Aion BRE generates the boxes designated as C code as well as the `ca.aion.j2aion.*` and `ca.aion.<nameaionapp>.*` packages.



The `<nameaionapp>jni.dll` (compiled C code) directly calls the Aion C-Code (dll) through a C-interface.

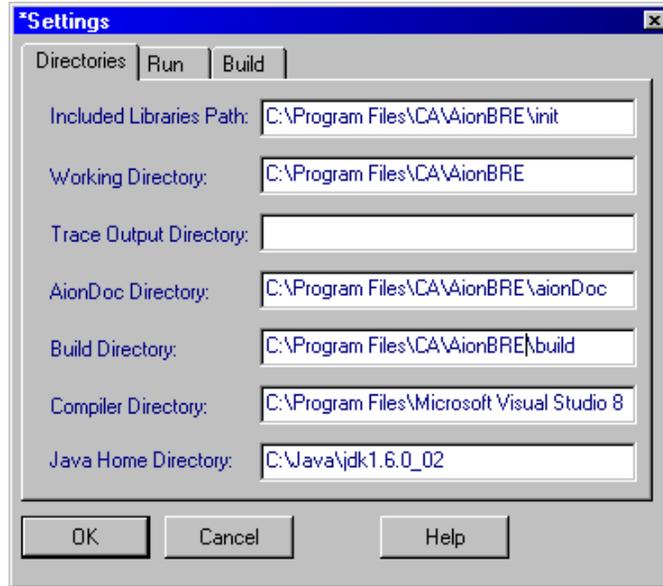
The `aionsession.jar` file is provided in the Aion BRE `\Build\rejava` folder. It contains two standard Java packages, `ca.aion.j2aion` and `ca.aion.j2aion.httpsession`, that provide concurrent thread-safe access to the Aion BRE component.

Other Java utilities provided in the Aion BRE `\Build\rejava` folder include the `log4j` package (see preceding diagram). This package contains logging facilities that are helpful in debugging the Java client application, and it is required when running the Java client application.

For more details on how the Java client application can be written, see *Java Application Programming: The Basics* and *Java Servlet Programming Considerations*.

Set Up the Environment

In the File Settings dialog, the Java Home Directory should point to the Java Development Kit (JDK).



The Java Virtual Machine (JVM) only loads native libraries from the library path. This library path is OS-specific. For Windows, the library path is added to the PATH environment variable.

Example:

C:\Program Files\CA\AionBRE\userLibs

Create an Aion BRE component Using the Java Interface

In this section we will create a simple Aion BRE application that is called from a Java client (see the HelloJavaWorld .app file in the examples/java/hellojavaworld folder provided with CA Aion BRE).

Code an Exported Class

We construct the classes whose public methods will define the interface of the Aion BRE component. The HelloJavaWorld example consists of the following elements:

- A class, for example, AionWorld.
 - Note:** In order to generate valid public class names in Java, the name of every interface class in Aion must begin with an upper case.
- An attribute, `f_message`. In the example, this attribute has an initial value of "Hello java world."
- Public get and set accessor methods.
 - Note:** To insure correct construction of the accessor methods it is advisable to use the Create Accessor wizard from the Create Accessor option on the Logic menu.

Export the class by checking Export on the Properties tab of the Class editor.

Note: The Export Properties button is not relevant for the Java interface layer.

Build the Aion BRE Application

To build the Aion BRE application, select the Java interface layer from the Build Directives tab of the Properties dialog of the library. If we have the "." included on the load library path, the java application loads the native libraries from the current folder. When we define an Executable directory on the Directories tab of the Library properties, CA Aion BRE copies all the required files to this folder. For this example, set the Executable directory to "..\exe". We will run the java application from this location.

Note: The Aion BRE application has to be compiled on the same type of platform as it will be executed.

File	Source	Function
aionsession.jar	Provided	Common behavior required by the Java client application
hellojavaworld.dll (appname.dll)	Generated	The compiled <aion? application
hellojavaworld.jar (appname.jar)	Generated	Java file containing the proxy classes for the Aion BRE exported classes
hellojavaworldjni.dll (appnamejni.dll)	Generated	Compiled C wrapper of the Aion BRE component; handles interface calls between Java client and Aion server

File	Source	Function
log4j*.jar	Provided by Apache	Java package for debugging the Java application
rejunn.dll; relsysnn.dll; resysnn.dll; reutilnn.dll	Provided	Aion BRE runtime dlls.

When deploying the Aion BRE server, the location of these dlls must be on the system's PATH variable.

More Information:

[The Basics of Java Application Programming](#) (see page 369)

Write and Compile the Java Client

Now create the Java application, for example, "MyJavaTest.java" in the executable directory. The Java client may consist of a single class whose main() method needs to perform primarily functions:

- Creating an instance of the exported Aion BRE application class (AionWorld).
- Calling an exported method of the class through the proper accessor method.

Note: For readability it is recommended that the Java client application imports the aionsession.jar and <nameaionapp>.jar files into the Java source application:

A simple Java client application can consist of the following code:

```
//importing the necessary classes makes the code more readable
import ca.aion.j2aion.AionSession;
import ca.aion.hellojavaworld.AionWorld;
public class MyJavaTest {
    public static void main(String[] args){
        //sent a message to the default output console.
        System.out.println("Starting java application");
        //If the ca.aion.j2aion.Aionsession and ca.aion.hellojavaworld packages are
        // are not explicitly imported, it is necessary to reference them in the
        // following statements.
        //Create an instance in the AionWorld class.
        AionWorld anAionWorldObject = new AionWorld(AionSession.getInstance( ));
        System.out.println(anAionWorldObject.getMessage( ));
    }
}
```

To compile the Java application, include the proper classpath in the compile:

```
javac -classpath .;aionsession.jar;hellojavaworld.jar MyJavaTest.java
```

where `hellojavaworld.jar` is the name of the jar file of your application (containing the proxy classes), and `MyJavaTest` is the name of the Java class to be compiled.

Note: If the Java application does not compile, ensure that the following conditions are true in your environment:

- `javac` and `javah` are present on the system path.
- the `.java` extension is specified on the name of the Java class to be compiled.
- appropriate Java case sensitivity is observed.

Test the Java Interface

Test the Java client through the following command prompt:

```
java -cp .;aionsession.jar;log4j-*.jar;hellojavaworld.jar MyJavaTest
```

where `MyJavaTest` is the name of the compiled Java class.

Note: The `log4j` package is required when running the Java application.

Running the `HelloJavaWorld` example should yield the following output written to the DOS prompt

```
Starting java application
Hello java world
```

The Basics of Java Application Programming

By including the generated `<nameaionapp>.jar` file in your Java program, you can access the exposed methods of the Aion BRE exported classes. However, there are programming considerations that must be implemented in the Java program to call an Aion knowledge base.

The essential lesson to know about calling an Aion server from a Java program is that the Java application must obtain a session object of type `AionSession` in order to bind the Aion BRE objects. Every Aion BRE object has to be created with a reference to an `AionSession` object.

The `aionsession.jar` file provides the means by which to create an `AionSession` under different environments. For standalone testing the Java programmer can create a simple `AionSession` object. (See the section `Aion Deployment on the Web` for steps in creating an `AionSession` under a web.) The following Java code is typical for how an `AionSession` can be created in a standalone Java application.

```
ca.aion.j2aion.AionSession myDefinedSession =  
ca.aion.j2aion.AionSession.getInstance();
```

Once an `AionSession` object is obtained, this object is used in the constructor of the proxy class (`AionWorld`).

```
ca.aion.hellojavaworld.AionWorld anAionWorldObject = new  
ca.aion.hellojavaworld.AionWorld(myDefinedSession);
```

In the preceding code, `AionWorld` is an exported class from the `Aion BRE` application, which is replicated as a proxy class in the `hellojavaworld.jar` file.

Once an object is created in the Java program, the exposed methods of the `Aion BRE` class can be called through that instance.

Example:

```
anAionWorldObject.setMessage("Hello Aion world");
```

where `setMessage()` is a public method of `anAionWorldObject`.

Note: References to non-exported classes in the Java code will cause compile errors.

More Information:

[Aion BRE Deployment on the Web](#) (see page 376)

Java Objects

The `AionSession` class provides the following functionality:

Type	Method	Explanation
<code>AionSession</code>	<code>getInstance()</code>	Factory method; returns a new <code>AionSession</code> based on the calling thread
<code>AionSession</code>	<code>getInstance(java.lang.Object sessionIdentifier)</code>	Overloaded Factory method; returns a new <code>AionSession</code> for a <code>sessionIdentifier</code> . The method can be used to create multi-threaded applications by binding the

Type	Method	Explanation
		AionSession to a thread Group. A socket client can also serve as a sessionIdentifier.
double	getVersionNumber()	Show the version number of the AionSession package
void	terminate()	Terminate an AionSession and force Aion to remove all allocated memory for a particular session

The AionObject provides the following methods:

Type	Method	Explanation
void	delete()	Delete the AionObject from the AionSession and, if appropriate, delete the associated Object in Aion, see Java Object Management.

More Information:

[Java Object Management](#) (see page 373)

Data Conversions and Exception Handling

The Java Interface Layer supports the following conversion from Aion BRE data types to Java data types.

Aion BRE Data Type	Java Data Type
Boolean	boolean
Integer	int
Real	double
String	String
(an instance of) <ExportedClass>*	(an instance of) <ExportedClass>
Date	Java Calendar
Time	Java Calendar
List of String	String[]
List of integer	int[]

Aion BRE Data Type	Java Data Type
List of real	double[]
List of date	Calendar[]
List of time	Calendar[]
List of Boolean	boolean[]
List of (instances of) <ExportedClass>	(instances in)<ExportedClass>[]

* Name of the exported class.

Exceptions are thrown by the Aion program during data conversion for Unknown and Null values, or when a constant violation occurs. These exceptions are subclasses of AionException, which is provided in the ca.aion.j2aion package.

- When an Aion attribute is unknown, an AttributeUnknownException is thrown.
- When an Aion attribute is Null, an AttributeNullException is thrown.
- When an assignment violates a constraint, an AttributeConstaintException is thrown.

You can catch these exceptions in the Java programming with standard exception handling:

```
try{
    myObject.getAttribute();
} catch (AttributeUnknownException aue) {
    // the Attribute is unknown in Aion.
}
```

Java Object Management

Aion BRE programmers are accustomed to using a class's `delete()` method to clear instances in Aion after they are no longer needed. Java favors automatic garbage collection, and therefore does not provide a corresponding method. However, the Java garbage collector will not remove objects outside its Java Virtual Machine (JVM). Moreover, there are situations in which it will be desirable for the Java program to delete instances of its proxy classes. For example,

- The instance of the proxy class is created and the Aion BRE object is created behind the scenes. Responsibility for object management in this case falls upon the Java programmer. The Java programmer must delete the proxy instance, which should delete its associated object in Aion.
- Aion internally creates an instance of an exported class (for example, through class containment), and a bit later the Java application asks for all instances of that class. Aion returns handles to the generated Java instances and the proxy class is instantiated on the fly (lazy instantiation). The Java environment can make inquiries of the newly created instances of the proxy class. The question is now, Who is responsible for the Aion created instance? Aion does not know that the outside world created a reference to the instance. The Java application does not know if Aion would like to keep the object. In this case, it is possible for Java to delete the instance without deleting the associated Aion instance.

In general, the philosophy should be that the creator of the object has the responsibility of deleting it:

Object Creator	Responsibility for Object Management
Aion creates object	Aion deletes object
Java creates object	Java deletes object
Aion proxy is created dynamically	Java object can be deleted; however the associated Aion BRE object will not be deleted.

Note: The attempt to reference an instance of a proxy class for which the associated object in Aion has been deleted will throw a `NullPointerException` in Java.

Supports Backward Chaining

A typical model for using inferencing systems is based on the traditional paradigm of "backward chaining": the application returns to the system user to obtain specific information on an as-needed basis. This model helps to focus the application in obtaining just that data that is required from the user. In standalone Aion BRE applications, this model can be directly supported by invoking backward chaining as an inferencing strategy and raising a query to the user when an unknown attribute is encountered during chaining. However, support for this model involves more complex issues when the Aion BRE application is accessed through a client application. With other interface layers, for example, the C interface layer, the Aion BRE application has to terminate backward chaining, keep the state of objects' attribute values and reinitiate the backward chain when new information is provided.

The Java interface layer allows the Aion BRE application to keep the state of the inference engine itself when returning to the user. Thus, the backward chain can resume for the point at which it was interrupted when processing returns from user. However, special programming considerations must be initiated on both the Aion and Java program sides in order to support backward chaining. This section highlights the basic principles for this support.

Note: The basic principles for supporting backward chaining in a client/server environment can be *implemented* with different techniques. The strategy illustrated here rests on the fact that Aion will throw an `AttributeUnknownException` in the calling Java client when it has failed to find a value for the goal.

Supporting backward chaining will typically involve the following principles:

- The Java client must call a public method on an Aion BRE application class that pursues the goal through backward chaining. This method must return an indication that the goal has been achieved or that additional information is needed by the Aion BRE application. Invoking this should be performed within a loop that exists only when the goal has been achieved.

Note: In a Java client that employs the strategy of detecting an `AttributeUnknownException`, invocation of the inferencing must be performed within the context of a try/catch block.

Example:

Where `evaluatePricingTier()` is an Aion method of the Java proxy `f_evaluator` that pursues and returns a goal:

```
for (;;) {
    try {
        l_pricingTier = f_evaluator.evaluatePricingTier();
        // no more unknown exceptions. Value is known.
        break;
    } catch (AttributeUnknownException aue) {
        askUserQuestion();
    }
}
...
}
```

Note: The unknown attribute on the Java side is the goal of backward chain, `l_pricingTier`, not the particular attribute that is discovered to be unknown on the Aion side (see the following third principle).

- The Aion BRE application must employ the `InferBegin()/InferEnd()` construction in order to preserve the state of the inference method throughout the process. The `InferBegin()` may be invoked in a begin-processing method that merely posts the rules and which is invoked only once by the Java client prior to its pursuing the goal in the inferencing loop (see previous principle). The `InferEnd()` is called when the Aion BRE application detects that the goal has been achieved.
- The Aion BRE application must also have a way recording the attribute that the inference engine discovers is unknown and for making this information available to the Java client. The Aion BRE application must support a method that the Java client can call to pass back to the Java client information about the attribute to be queried (such as the format of questions and, if relevant, constraints on the response). The Aion BRE application must finally support another method that receives the user's response back from the Java client, populates the appropriate object's attribute (based on the saved `attributepointer`), and resumes inferencing.

Note: These requirements are typically accomplished by using the Aion `attributepointer` type and other meta-programming capabilities.

Aion BRE Deployment on the Web

A value of Java Interface Layer is that permits Aion BRE rule servers to be deployed on the web and accessed by Java clients running in a multi-threaded environment. This section will address specific issues related to web deployment.

Roles and Responsibilities

A web development project can be expected to involve several distinct programming roles requiring much different skills: the web designer, the Aion programmer, and the Java (servlet) programmer. For example, the web designer need not know anything about knowledge base system construction. Similarly, it is not expected that the Aion programmer will necessarily be an expert on Java servlet programming. The Aion programmer just needs to publish the interface of the Aion BRE application. It is the Java servlet programmer who knows how to write the controller application (servlet) that invokes the Aion BRE application.

The following table summaries the responsibilities and typical tools of the required roles.

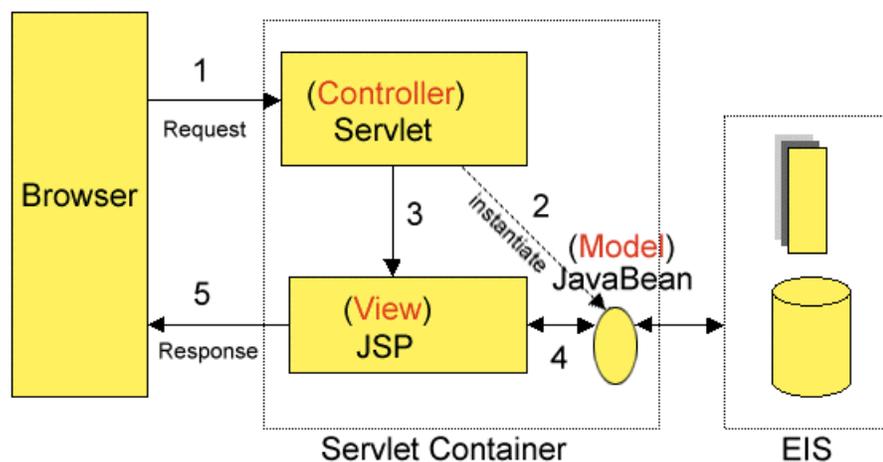
Role	Responsibilities	Tools
Java programmer	Procedural logic, test, interacting with the Internet, thread architecture and session management, database connectivity.	JDK, Swing (for GUI), JDBC
Graphical User Interface designer	Presentation on the Web, usability, documentation, training	FrontPage, Adobe, DreamWeaver, JSP, cognitive psychology
Knowledge Engineer	Knowledge acquisition, analysis and codification of business logic, understanding inferencing, object-oriented analysis and design.	Interviewing and ethnographic analysis, logic representations, Aion
Domain Expert	Defining vocabulary of the domain, articulating the business logic.	Dynamic Rule Manager
Deployment Engineer	Manage the Web environment	Weblogic or WebSphere

Note: This separation of roles is consistent with the basic philosophy of component-based development, namely that each component should only be required to know how to do tasks in its area of responsibility (the principle of encapsulation of responsibility). The J2EE architecture is found upon this principle as well; for example, JavaServerPages (JSP) were developed for graphic user interface designers so that they did not have to know Servlet programming to develop GUIs.

Servlet Technology

The recommended architecture for invoking an Aion server within a web environment is known as Model 2 Model-View-Controller pattern, a form of the classic Model-View-Controller (MVC) pattern. This model is the recommended best practice by SUN. Simpler models are possible, for example, having a single Servlet or Java Server Page (JSP) to handle requests and to generate the HTML page. This model is suitable for simple applications but is not desirable for large applications in which we would expect Aion BRE to be used.

Having Aion generate an HTML page is not recommended for both performance reasons and design reasons. We do not recommend that a business model contain view-specific code.



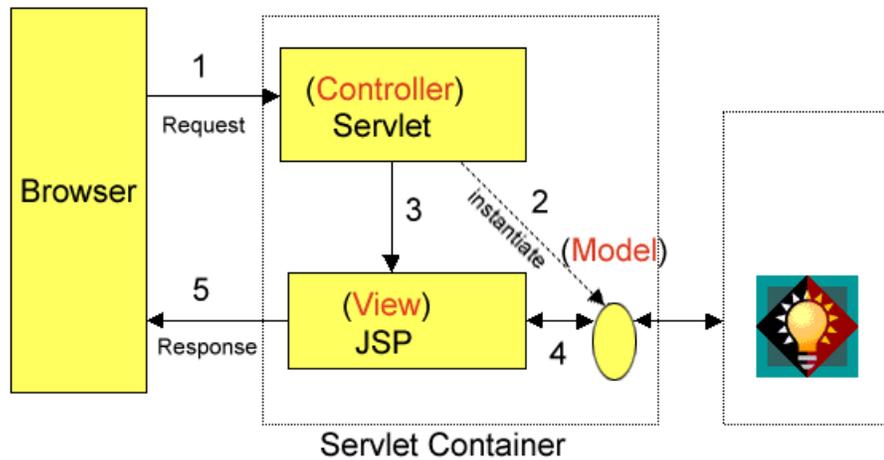
(Source:
<http://developer.java.sun.com/developer/onlineTraining/JSPIntro/contents.html#JSPIntro4>)

In this model, processing is clearly divided between presentation and front components. The latter alone handles all HTTP requests. These front components are responsible for creating beans used by the presentation components. One advantage of this model is that the front components provide a single point of entry into the application, thus making the management of application state, security, and presentation uniform and easier to maintain.

For more information about Model 2, visit:

<http://java.sun.com/developer/onlineTraining/JSPIntro/>

The corresponding structure for invoking an Aion BRE application is a variation on this structure:



Aion Session Initiation and Object Creation

Under this model, the Servlet (Controller) creates an `AionSession` using a `sessionIdentifier`. For the web, an `httpSession` serves as just such a `sessionIdentifier`. The Java programmer, when working within a J2EE environment, can use the `getInstance()` method of `AionHttpSession` to obtain an `AionSession` instance.

Example:

```
AionSession myDefinedSession = AionHttpSession.getInstance(httpSession)
```

Note: When working with a J2EE environment, the Java programmer should not use `AionSession.getInstance()`.

Note: A powerful feature of the Aion solution is that it can accommodate different kinds of sessions. Aion can be bound to a socket, to a thread group, to an http session (servlets managed by a web server as described in this section), or to an XML SAX parser.

The Servlet must create instances of the proxy classes using this session object.

```
MyClass myClass = new MyClass(myDefinedSession);
```

As a convenience, these instances can be created by invoking the `Instantiate()` method of the `AionBean` class. Once a proxy instance is created, the public methods of the corresponding class in Aion are accessible to the Servlet.

More Information:

[Create Aion BRE Objects for the Web](#) (see page 380)

Aion Session Termination

The `ca.aion.j2aion.httpaionsession` package is written to be compliant with the Servlet 2.2 API, which is part of the J2EE framework. If this API is available, the `AionSession` is automatically terminated when the Java program ceases execution or when there is an abnormal failure.

If this Servlet API is not available, the Java developer must explicitly call the `terminate()` method on the session object regardless of whether the session terminates normally or abnormally. When the session terminates, the final statement must be

```
myDefinedSession.terminate( )
```

Java Servlet Programming Considerations

The following sections describe considerations that the Java programmer should be aware when calling an Aion knowledge base with the web environment.

Create Aion BRE Objects for the Web

Remember that an Aion BRE object can only be created based upon an AionSession. The `ca.aion.j2aion.httpsession` package provides alternative means of creating AionSessions and proxy objects besides that provided by the `ca.aionj2aion` package (see Java Application Programming: The Basics). The mechanisms provided by this package can be used to create AionSessions under a web environment.

The `ca.aion.j2aion.httpsession` package provides the AionHttpSession class. AionHttpSession is derived from the AionSession class.

- Invoke `AionHttpSession.getInstance(javax.servlet.http.HttpSession sessionIdentifier)`

Use the AionBeans Class

The `ca.aion.j2aion.httpsession` package also makes available the class AionBeans, which is used for accessing Aion BRE objects as Java beans. Beans can be created through one of the overloaded `instantiate()` methods:

Type	Method	Explanation
AionObject	<code>instantiate(java.lang.ClassLoader cls, AionSession aionSession, java.lang.String className, java.lang.String beanName)</code>	Create an aionObject as a bean in the specified session
AionObject	<code>instantiate(java.lang.Classloader cls, javax.servlet.http.HttpSession httpSession, java.lang.String beanName)</code>	Convenient way to create an aionSession based on the httpSession and instantiate an object within that session

Creating an AionSession by using the `AionBeans.instantiate()` method is done in the following code:

```
anAionObject = AionBeans.instantiate(null, httpSession,
    AnAionSession.AionWorld.getName( ), aionSessionObjectName);
```

Note: A Servlet has specific properties that enable it to terminate an AionSession automatically when an `httpSession` is invalidated. When an `httpSession` terminates, the `aionSession.terminate()` method will be automatically called for cleaning up any remaining Aion instances.

The AionBeans class also provides the following helper method to call Aion accessors:

Type	Method	Explanation
void	setProperties(java.lang.Object bean, javax.servlet.httpServletRequest request) throws javax.servlet.ServletException	Maps http parameters directly to the Aion set accessors

This utility makes use of bean introspection to execute the accessor method on object. Based on its the input parameter, the method determines whether the object has an appropriate accessor method, and if so, calls the accessor to set the value of an attribute of the object. For example, an http parameter `http://server/yourServlet?age=22` may be retrieved when a form having an input field named "age" is submitted to a Servlet. Passing this parameter to the `setProperties()` method causes the Aion method `setAge(22)` to be invoked if a `setAge()` accessor is present on the object.

It is assumed that every property has a set accessor at the bean. If the set accessor is missing on the bean, nothing is set. The supported types in this helper method are:

- Java boolean → mapped to Aion boolean
- Java int → mapped to Aion integer
- Java double → mapped to Aion real
- Java String → mapped to Aion string.

Note: Because introspection introduces additional overhead, you might want to program the set accessor directly.

For performance reasons, it is also recommended to pass a date value from a form by three integers `day_int`, `month_int` and `year_int` and let Aion create the datatype based on these parameters: `myDate.setDate(day_int, month_int, year_int)`, instead of creating the Java Calendar object and passing the calendar object into Aion. Similar considerations apply to setting Time.

Complex types like addresses, for example `setAddress(newAddress)`, must be programmed with form helper classes.

Java Server Page (JSP) Programming

In the MVC architecture, the Servlet is responsible for creating objects and for directing the next JSP page. Using a hidden value on a form, the JSP programmer can specify the name of the Aion BRE object.

Exempt of the create_myaioworld.jsp:

```
<form method="POST" action="../controllerservlet">
  <input name="anAionWorldName" value="myAionWorld" type="hidden">
  <input type="submit" value="Create Session" name="action">
</form>
```

Example of the corresponding Servlet code:

```
AionWorld aionWorld = null;
String aionWorldName = request.getParameter("anAionWorldName");
String action = (String) request.getParameter("action");
if (action.equalsIgnoreCase("create session")) {
    session = request.getSession(true);
    aionWorld = (AionWorld) AionSessionBean.instantiate(null,
        HttpSession, AionWorld.class.getName(), aionWorldName);
    session.setAttribute(aionWorldName, aionWorld);
    forwardJSP(request, response, "edit_aionworld.jsp");
}
```

Other JSP pages can refer to the Aion BRE object by using the jsp:useBean construction:

```
<!-- retrieve the Aion BRE object with the name handle as specified in
the hidden field on page create_myaioworld.jsp -->
<jsp:useBean id="myAionWorld" class="ca.aion.myaioworld.AionWorld"
scope="session" />
```

An attribute value of the Aion BRE object can be shown by:

```
<jsp:getProperty name="myAionWorld" property="message" />
```

Thread Management

Deploying an Aion server on the web raises questions about thread management and safety. This section will draw upon the discussion in the preceding section to present a more detailed answer to these issues (see Servlet Technology).

Note: The following discussion focuses on user-level (application) threads rather than kernel-level (operating system) threads.

General Definitions

A **thread** is a sequence of instructions, executed within the context of a process, that services one individual user or a particular service request. A thread allows a program to know which user is being served as the program gets re-entered by different users. Threads are “managed” by the environment (for example, by MTS) or by the application (Java). There are several general approaches to thread management.

Single-threading

The ability of a program to process a single thread for a single user at a time. Other threads needing access to the resources of the program must wait their turn until the executing thread has completed. For example, the inference engine in Aion is single-threaded. It does not permit multiple users performing different inferences over the same set of objects.

Multi-threading

The ability of a program to maintain multiple threads of control through the same code without incurring interference of the threads with each other. Some programming languages, such as Java, support multithreading with a special programming mechanism called synchronization. Synchronization allows one thread to place a lock on an object during code execution. In Java threads are synchronized to objects that implement the Synchronize interface. A requirement of a synchronization scheme allows a thread to recognize its own lock if the thread has to be reentered.

Thread safety

Preventing unwanted interaction between threads, such as accessing each other's data.

While Aion is thread safe, the Aion language does not support multithreading. However, this fact does not prevent Aion servers from functioning efficiently and safely in a multi-threaded environment-see Support for Concurrency and Session Safety.

Support for Concurrency and Session Safety

For deployment on the web, Aion servers must be usable with a multi-threaded Java environment. To accomplish this, Aion supports session safety and concurrency.

Note: CA Aion BRE's solution is to synchronize on the `aionSession` object.

Recall that invoking an Aion knowledge base from a Java application requires that the Java application create a session, called an `AionSession` object. When deploying an Aion knowledge base on the web, an `aionSession` must be created based on an object within the environment, for example, an `Http` session. By synchronizing on this object, this architecture supports safe, concurrent execution within a multithreaded environment.

This is a standard architecture, under which it is up to the developer to determine how the code is to be locked. By default the `AionSession.getInstance()` will create an `aionSession` based on the current thread. If 1000 threads are created, there will be 1000 `aionSessions` and these `aionSessions` can be called *concurrently*.

(By analogy, consider the hashtable. A hashtable is an object in Java for which only one thread can mutate the hashtable at a given time. The hashtable operations are synchronized and are thread safe. The same holds true for Aion: only one thread can effect an `aionObject` at one given time. An Aion session is synchronized on the `aionSession` object.)

Because the synchronized `AionSession` object is a parameter in the creation of proxy class instances, every Java method is wrapped in a synchronized code block. This strategy disallows multiple threads from accessing the same Aion session at the same time. Multiple Aion BRE sessions can be accessed concurrently within a multithreading environment.

Each Aion session has its own symbol table, or set of objects that it uses. The symbol table represents the data used within that particular server instance. The processing logic is shared amongst the instances. Thus, a second thread could run, using its particular symbol table, while another is executing.

Note: Creating an Aion session using an `http` session that already supports an `aionSession` returns the same `aionSession`. In general creating an `AionSession` based on the same `sessionIdentifier` will return the same `aionSession`. This strategy allows a thread to reenter its own session without being prohibited by its own lock.

Resource Load Issues

How is the Aion BRE application loaded at runtime and what resources does it use in a multithreaded environment?

The Aion executable, the .dll, represents the code space. The .dll is only loaded once per process (not per thread) by the operating system. Additional memory is required only for each additional aionSession. This memory of course, is used to allocate dynamic data such as symbol tables, object instances, class variables, and constants. The aionSession acts as a separate instance of the JVM. All of the threads simply point to the executable code space (from within their respective symbol table). Thus, many instances can be spawned from the single instance of the executable.

Additional Information

This section provides sources for more information about Java and Java Servlet technology. An excellent general source of information about Java, where you can get white papers and links to other sites that cover technical discussions and training material, is <http://www.jguru.com>

The following sites address specific topics:

- For basic information on using Java (for example, how to compile Java code), visit:
<http://java.sun.com/products/servlet/index.jsp>
- To get started with servlet technology, visit:
<http://java.sun.com/products/servlet/index.html>
<http://developer.java.sun.com/developer/onlineTraining/Servlets/Fundamentals/index.html>
- To learn more about the Servlet+JSP engine, Tomcat, visit:
<http://jakarta.apache.org/tomcat/index.html>
- To learn more about using the log4j package to debug your Java applications, visit:
<http://jakarta.apache.org/log4j/docs/index.html>

Chapter 16: Generate and Use COM Components

This chapter discusses how you can use CA Aion BRE to generate Microsoft Component Object Model (COM) objects, as well as access and use existing COM objects from within CA Aion BRE. COM specifies how to build software components that can be dynamically interchanged. COM provides the binary standard that clients and these components (including libraries, applications, and controls) use to interact with each other. By implementing COM, a software product implements its services as one or more COM objects. COM objects provide services via methods that are grouped into interfaces that can never change. All interaction is through interface references. COM is a *Model*, which is implemented as a DLL or EXE. It is language independent (and location independent with DCOM).

OLE (Object Linking and Embedding) was the first COM system and refers to the technology used to create compound documents. OLE is a persistent storage mechanism that is used to physically store information about COM objects that a client (container) application uses.

This section contains the following topics:

[Automation](#) (see page 388)

[Aion BRE and COM](#) (see page 389)

[Object Generation Overview](#) (see page 390)

[Generate COM or ActiveX Objects](#) (see page 392)

[Use Aion as an Automation Client or Server](#) (see page 394)

[AutoLib Example](#) (see page 395)

[COM Interface Server Side Example](#) (see page 397)

Automation

Software applications use Automation (formerly called OLE Automation) to expose their services, making them programmable to scripting tools and other applications. The IDispatch interface is used to expose these services.

Automation server

An *Automation server* is a COM component that implements the IDispatch interface. An Automation server can be an in-process server (the same process as a client and implemented as a DLL) or an out-of-process server (a different process as a client and implemented as an EXE). Furthermore, the server can be *local* (running on the same machine as the client) or *remote* (running on a machine other than the client). The section Generating COM or ActiveX Objects discusses using CA Aion BRE to generate Automation Servers.

Automation controller

An *Automation controller* is a COM client that communicates with the Automation server through the server's IDispatch interface. The section Using Aion as an Automation Client discusses using Aion applications as Automation Controllers.

ActiveX

ActiveX is a broad set of COM client and server technologies that can be used to develop interactive content for applications used on the World Wide Web.

These technologies include ActiveX controls (formerly called OLE controls), ActiveX documents, Active Scripting, and ActiveX Database Objects (ADO). ActiveX documents enable users to view non-HTML documents, such as Microsoft Excel or Word files, through a Web browser. Active Scripting is used to control the integrated behavior of several ActiveX controls from the browser or the server. ADO is used to write client applications that access and manipulate data in a database server.

DCOM

DCOM is distributed COM.

All COM-conforming objects automatically inherit distributed properties. All COM objects reside in local servers and provide marshaling proxies and stubs that can be accessed remotely without having to modify the binary image.

MTS

The Microsoft Transaction Server environment (MTS) is a powerful *runtime* environment for hosting COM components. MTS simplifies development and deployment of server-centric COM applications, and is fully integrated with IIS and ASP for easier internet/intranet application development. MTS provides automatic transaction support, is scalable and secure, and is ideal for three-tier systems.

COM+

COM+ is the latest generation of the COM standard; it combines enhancements to COM with a new version of the Microsoft Transaction Server (MTS). It provides many new services, and eliminates unsafe manual life-cycle management of objects. These services are made available through a container, called the *application*, for managing and executing COM and COM+ components. An application is a group of COM and COM+ components that perform related functions.

Aion BRE and COM

CA Aion BRE uses two Component Object Model-based technologies: Automation and ActiveX controls. The Component Object Model provides the following:

- The interface that a client uses to communicate with a service provider
- An architecture that object classes use to support several interfaces
- A memory management system to allocate and free memory between a client and an object
- Error checking and reporting
- Transparent communication of objects
- Identification and loading of DLLs

Aion BRE and COM+

Currently Aion BRE supports the following COM+ services.

- Role-based security to administratively construct an authorization policy for an application.
- Transaction services.

More Information

[COM+ Services: Server Side](#) (see page 398)

Object Generation Overview

CA Aion BRE can be used to generate lightweight COM objects (non-GUI Automation Servers) or can be used as an Automation Client. CA Aion BRE also can generate ActiveX controls or can be used as an ActiveX container.

Library Interface Layer	Generates
ActiveX	ActiveX control (OCX)
COM (In Process)	In-process Automation server (DLL)
COM (Out of Process)	Out-of-process Automation server (EXE)
MTS	Microsoft Transaction Server (DLL)
MVS COM	In-process Automation server (DLL)

Set Up the Environment

To build Aion applications with the COM interface under Visual Studio 6.0, you must have access to Microsoft Visual Studio. The INCLUDE path in the environment variable should specify: *vsdir\VC98\atl\include*, where *vsdir* is the Microsoft Visual Studio installation directory (for example, C:\Program Files\Microsoft Visual Studio). Alternatively, you can also specify this directory in the Included Libraries Path field in the Settings dialog.

Note: This set-up is not necessary when building Aion applications with the COM interface under Microsoft Visual Studio 7.0 (Microsoft Visual Studio .NET). CA Aion BRE automatically sets the necessary paths at build time.

COM Object Generation

You can use Aion to generate a COM object that is In-Process (a DLL), or Out-of-Process (an EXE). The COM generation process uses the Active Template Library (ATL). Aion does not need to generate proxy-stub code because COM provides a universal marshaller (oleaut32.dll).

Note: Aion automatically generates the IDispatch interface to support VBScript and JavaScript.

In-Process COM objects (servers) can be contained and aggregated. Generated COM objects support a subset of automation data types. Supported data types are discussed later in this chapter.

You can generate COM objects for non-graphical server objects. For GUI objects, use the ActiveX Interface Layer.

ActiveX Object Generation

CA Aion BRE supports the generation of ActiveX controls (OCXs). Generated ActiveX controls are In-Process (DLLs) which can support User Interface elements. Generated ActiveX objects support a subset of automation data types.

Note: For performance reasons, especially over a network, it is advantageous to create non-graphical rule objects as lightweight COM objects (non-GUI automation servers) rather than as ActiveX controls.

Aion-generated ActiveX controls can be used in ActiveX containers such as Visual Basic and Internet Explorer. ActiveX controls use Automation to perform event firing and Get/Set methods and Properties. Without ActiveX controls, the end user cannot respond to event firing and Get/Set ambient properties at runtime.

Aion uses the Class Properties dialog to export a class as an ActiveX control.

MTS Object Generation

Generated MTS objects are In-Process (DLLs). Aion implements MTS objects using the Active Template Library (ATL). The generated server is a non-aggregate server and supports the Apartment threading model (Single Threaded Apartment or STA).

Note: If you are planning to deploy a component under MTS, you cannot use the COM (In-Process) Interface Layer. Even though both are DLLs, in this circumstance you must use the MTS Interface Layer.

COM+ Object Generation

COM+ supports only In-process components. Currently, there is no difference between generating a COM object and generating a COM+ object. There is no special interface layer for COM+. The COM In-process interface layer should be chosen at build-time for COM+ components.

MVS COM Object Generation

Generated MVS COM objects are In-Process (DLLs) which can be used in a client/server configuration between a mainframe and a PC. MVS COM objects are built on the PC to generate a proxy (implemented as a COM object) that speaks TCP/IP to a mainframe server application, and COM to the client application.

In addition to all public classes in the application, the proxy (DLL) contains an automatically-generated class named *cc_connection*. This class contains methods used to communicate to the server application at runtime.

Generate COM or ActiveX Objects

When you generate Aion COM or ActiveX objects, you choose one or more Aion classes whose methods you want to expose. This process consists of the following steps:

- Choose which Aion classes to export as COM or ActiveX objects.
- Specify a COM or ActiveX Interface Layer for the application.
- Build the application to create an EXE and/or DLL(s).
- Register the generated object.
- If using COM objects in a distributed system, configure DCOM and then use it to distribute the objects.

Note: You do not need to include AutoLib or ActiveXLib to generate a COM or ActiveX object *unless* you are using COM server side interface support, which requires AutoLib. You do not need to include ComLib to generate a COM+ server. ComLib is required only for program-level access to and control of the COM+ services provided by the library.

For step-by-step procedures for generating in-process (DLL) and out-of-process (EXE) COM and ActiveX objects, see Generating COM or ActiveX Objects in the CA Aion BRE online help.

Data Type Support in Automation Servers

The following table lists which Aion data types are supported and the COM type to which they are mapped:

Aion	COM	Input	Output	Return
Array	SAFEARRAY	Yes	No	No
Boolean	VARIANT_BOOL	Yes	Yes	Yes
COMInterface	IDispatch	Yes	Yes	Yes
Date	DATE	Yes	Yes	Yes
Integer	Long	Yes	Yes	Yes
Real	Double	Yes	Yes	Yes
String	BSTR	Yes	Yes	Yes

Register the COM Object

Before a client can access the built COM object, it must be registered in the server machine's registry.

For step-by-step procedures for registering a COM object, see Registering the COM Object in the CA Aion BRE online help.

Verify Successful Registration

For step-by-step procedures for verifying that COM object registration was successful, see Verifying Successful Registration in the CA Aion BRE online help.

Configure DCOM

This section only applies if the COM objects are being accessed from a remote machine.

Test the DCOM Configuration

You can use the DCOM-enabled NetClip application to test the DCOM configuration. For more information, see:

<http://www.microsoft.com/com>

For step-by-step procedures for testing the DCOM configuration, see Testing the DCOM Configuration in the CA Aion BRE online help.

Depending on which operating systems the client and server machines are running, the testing procedure differs slightly.

COM+ Application Configuration

Configuration is an essential part of the development process for COM+ applications. How you configure an application will determine how the components behave when running.

Applications are defined and configured through the Component Services tool. Component Services is an administrative tool that is included in Windows 2003/XP. You can use the Component Services administrative tool to create new COM+ applications, add components to applications, and set the attributes for an application and its components.

Current COM generated components can be configured in a COM+ application. It is possible to mix COM and COM+ objects in the same application.

Note: Refer to the Microsoft Windows manuals for further information about Component Services.

Use Aion as an Automation Client or Server

When you use Aion as an Automation Client or Server, you use existing COM interfaces to create a new COM object class in an Aion application. You can also return and use pointers to COM interfaces in COM servers created from an Aion knowledge base.

This process consists of the following general steps:

- Browse through existing COM objects (including Type Libraries) to locate interfaces.
- Choose interfaces from which to create one or more new COM object classes.
- Generate the Methods and Properties for each new class.
- Use the generated methods in conjunction with Autolib methods.

Note: Aion can use any COM object, including those generated by C++, Visual Basic, Java, and Aion (as long as IDispatch is implemented).

Include COM Objects in Aion Applications

For step-by-step procedures for making COM objects accessible to your Aion application, see Including COM Objects in Aion Applications in the CA Aion BRE online help.

Data Type Support in Automation Clients

The following table lists the COM data types that Aion supports and the Aion types to which they are mapped:

COM	Aion	Input	Output	Return
BSTR	String	Yes	Yes	Yes
DATE	Date	Yes	No	No
Double	Real	Yes	Yes	Yes
IDispatch	COMInterface	Yes	Yes	Yes
Long	Integer	Yes	Yes	Yes
SAFEARRAY	Array	Yes	No	No
VARIANT*	_Datatype	Yes	Yes	Yes
VARIANT_BOOL	Boolean	Yes	Yes	Yes

*Only Boolean, Date, Integer, Real, and String are supported for the Variant data type.

AutoLib Example

Note: The following example is provided in the Example\COM\Animals directory on the Aion CD.

The Start() Method of client.app illustrates some of the methods used in AutoLib. The two COM objects, Dog and Cat, are implemented as .DLLs. The Dog object is created and called from the client app; however, the Dog object needs to create and call a Cat object. This example illustrates the use of callbacks between COM objects.

Start() Method in client.app

The following code is an example of the Start() method in client.app:

```
pDog = IDog.Create( )
if(pDog <> NULL)
then MessageBox("Dog created!", "CLIENT")
else MessageBox("Cannot create Dog! ", "CLIENT")
Abort( )
end
pDog->cc_CallCat(pDog.GetDispatch( ))
pDog->Delete( )
MessageBox("Goodbye!", "CLIENT")
```

The Create method is used to create a new instance of the object 'Dog' on the same machine. If the Create method creates an instance of DOG, a pointer to the Dog object (pDog) is returned.

The Dog's object CallCat method is then called.

Implement Callbacks Between COM Servers

In this example, the Dog instance creates a Cat instance and the Cat instance calls the Dog's Bark() method. That is, the instance of Cat created by Dog *calls* back to its creator to invoke a method on its creator while it is processing a method invoked on it from this same creator (Dog) instance. This situation is called a "callback".

The logic in Dog's CallCat() method, which has been invoked from the client, is straightforward:

```
IN dogid is integer
pCat = IMyCat.Create()
if (pCat <> NULL)
then MessageBox("Cat created!", "DOG")
else MessageBox("Cannot create cat!", "DOG")
    Abort()
end
pCat->cc_SetCallback(dogid)
pCat->Delete()
```

Cat's SetCallback() method employs the AttachDispatch() method, which insures that the existing instance of Dog is used:

```
IN arg1 is integer // dogid from client.app
pDog = IMyDog.Create(NULL, FLAG_COM_NONE, NULL)
pDog.AttachDispatch(arg1)
pDog.cc_Bark()
```

COM Interface Server Side Example

The preceding example illustrates retrieving and using pointers to an instance of a COM object on the client side, `pDog = Idog.Create()`. It is often necessary to return and use pointers to COM objects on the server side. To return a pointer to a COM object to the server side it is necessary to derive the class being created from the `COMShell` class in `AutoLib`. The `COMShell` class acts as a placeholder for COM classes that do not yet exist (at edit time). At runtime, `COMShell.Create()` instantiates the COM Interface from the program ID (`progid`) that is passed in. This means the COM server application needs to include `Autolib`.

Consider an Aion COM server application that exposes a `Person` interface, with a method `GetDog()`, and a `Dog` interface. `GetDog()` method retrieves an instance of a `Dog` COM object. (For later use in this example, we assume that the `Dog` interface exposes a `Bark()` method.) `IDog` must be defined as a class that is derived from `COMShell`.

Server Side

The following code is typical implementation of the `GetDog()` method on the server side (in `Person`):

```
var p is &IDog
var dDog is COMInterface
p = IDog.Create(NULL, "Comisrvr.Dog.1") // invokes COMShell.Create( ) with ProgID
    // or ClassID to find object in registry
dDog = p.GetDispatch( )
return dDog
```

This `GetDog()` method creates an instance of the `Dog` and returns a handle to the `Dog` interface.

Client Side

On the client side, you can import the COM object from its type library. In this case, you create an `IDog` object and an `IPerson` object. You can also generate a `cc_Bark()` method for the `IDog` class and a `cc_GetDog()` method in the `IPerson` class. You can then have the person create a dog instance and call `cc_bark()` on that instance of dog:

The following code executes this scenario:

```
var pPerson is pointer to IPerson
var pDog is pointer to IDog
var iDog COMInterface
pPerson = IPerson.Create( )
pDog = IDog.Create(Null, FLAG_COM_NONE)
iDog = pPerson->cc_GetDog( )
pDog->AttachDispatch(iDog)
pDog->cc_Bark( )
pDog->Release( )
pPerson->Release( )|
```

COM+ Services: Server Side

Many of the services provided by COM+ can be administratively configured for an application simply through the Component Services Administration Tool. However, there are some tasks that may need to be done or it might be desirable to do programmatically. COM+ provides a set of interfaces such as `ISecurityCallContext`, `IObjectContext`, `IContextState`, `IObjectControl`. A subset of these services is currently available to the Aion programmer through the `COMPlusObject` class that is provided by the `CompLib`.

To use the facilities of COM+ at a programming level within an Aion server, exported classes must be derived from the `COMPlusObject`.

Role-Based Security

Although an Aion generated COM+ component can be configured to use Role-Based Security with the Component Service Administration tool, programmers can also use `ISecurityCallContext` interface to determine whether security is enabled for the current call, and check role membership to determine whether a particular section of code is executed. The following methods of `COMPlusObject` support role-based security:

- `GetCallContext`
- `IsCallerRole`
- `IsSecurityEnabled`
- `IsUserInRole`

For more information on these methods, see the "CompLib" chapter of the CA Aion BRE online help.

Transaction Services

To use automatic transactions effectively, each transactional component should indicate that it has completed its work. When an object instance completes its task successfully, it should set its `Consistent` bit to `True` and its `Done` bit to `True` by calling the `SetComplete()` method of the `IObjectContext` interface. The following methods of `COMPlusObject` support transaction services:

- `GetObjectContext`
- `IsInTransaction`
- `SetAbort`
- `SetComplete`

For more information on these methods, see the "CompLib" chapter of the CA Aion BRE online help.

Chapter 17: Deploy Aion BRE Components as Web Services

The Web services environment is a framework for supporting electronic collaboration, including systems integration and business-to-business (B2B) communications. Web services promote interoperability. The ability of Aion BRE components to act as Web services simplifies the creation of collaboration points between different business processes.

This section contains the following topics:

[Program Aion BRE Components as Web Services](#) (see page 399)

[Do I Need To Install Apache Axis?](#) (see page 401)

[Code Generation for Web Service Deployment](#) (see page 409)

[Client Programming Considerations](#) (see page 410)

[Administering Aion-Based Web services](#) (see page 411)

[Additional Resources on Web Services](#) (see page 411)

Program Aion BRE Components as Web Services

When programming an Aion BRE component whose interface you wish to publish as a Web service, you should program to the standards and conventions of the Java interface layer.

As in the Java Interface Layer, exception throwing is supported.

More Information:

[Code an Exported Class](#) (see page 367)

Use Complex Data Types

The Web services interface supports passing complex types between the client and the component implementing the Web service. Complex data types are types that are defined as consisting of a collection of other complex or primitive data types. For example, a Person may be a complex data type consisting of a number of primitive data types of strings and integers (for Name and Age respectively) and complex data types (such as Address, which may consist of separate strings for Street, City, State, and more). In traditional object-oriented programming, complex data types are supported by defining the type as a class. To support complex data types in a Web services interface, construct the class defining the complex data type in the following manner:

- Define public methods of a class that will serve as the exported class for the Web services in terms of pointers to other classes that define your complex data types, for example, `processInfo(&InputData) : &Info`, where "InputData" is a complex data type (class) describing input to the `processInfo` method and "Info" is a complex data type (class) describing the information returned by the method.
- Ensure that the classes defining complex data types are exported.
- Define public accessor methods on each class defining a complex data type according to the Java standard for each data type that is part of the complex data type. The accessor methods may involve either primitive data types or further complex data types (that is, pointers to further classes defining complex data types).

Java inspection (provided by Apache Axis) constructs the definition of the complex data type that corresponds to the class definition when generating the WSDL.

Program Standards for Web Services

Standards must be kept in mind when programming for Web services deployment:

- Current Web services standards (WSDL) restrict publishing the methods of only a single class as Web services. Users who want to export methods from multiple classes must create a façade class that exposes the desired methods within a single class.
- Program to the standards of a stateless service. Do not expect the Aion BRE component to maintain state during successive invocations of its services.

- Because Apache Axis automatically sets attributes with public accessors to NULL when the object is first created, care must be taken to ensure that such an attribute has an Unknown status before it is used in an interface. In particular, this problem arises if the attribute is used in a decision table and has a domain interface member defined for it. It is recommended that before initiating inferencing, all goals are explicitly returned to the Unknown state:

```
_engine.goalMakeUnknown(->attributeName)
```

- A side effect of the preceding point is that if an attribute is of constrained type and the list of constrained values does not include the NULL value, then Axis causes a constraint violation when setting the attribute to NULL. Therefore, NULL must be included in all constraint lists of attributes with public accessors.

Do I Need To Install Apache Axis?

For users intending to use Aion BRE Web services on Windows or UNIX/Linux systems, Apache Axis must be available. This is provided when the CA Aion version of Portal is installed. However, if CA Aion BRE was installed by itself (without Rule Manager), the following additional steps are required:

To install Apache Axis

1. Install an Application Server that is compatible with Apache Axis 1.4 or higher. Axis recommends Apache Tomcat (see <http://tomcat.apache.org/>).
2. Install Apache Axis 1.4 or higher in the Application Server webapps directory (see the *Axis Installation Guide* at <http://ws.apache.org/axis/java/install.html>) Configure Apache Tomcat as follows:

Enable monitoring of Axis lib directory for class reloading as new Aion BRE services are built and deployed. Using the Tomcat Administration console (<http://localhost:CA Portal/admin>), locate the Context entry for the Axis servlet (/axis) and set the following properties:

- Cross Context = True
- Reloadable = True Check Interval = 3

For Microsoft Windows installations, add the Aion BRE \userlibs folder to the Windows system PATH in the Tomcat startup scripts:

- Create the Tomcat environment setup script (*Tomcat_install_dir\bin\setenv.bat*). This batch file is automatically executed by the Tomcat start-up scripts on Windows systems.
- In the file *bin\setenv.bat* enter the following commands:

```
set PATH=%PATH%;<AionBRE_install_dir>\userlibsAionBRE_install_dir
```

This is the full directory path where CA Aion BRE is installed on the Windows platform.

For UNIX/Linux installations, add the Aion BRE userlibs directory to the system library path (LIBPATH, SHLIB_PATH or LD_LIBRARY_PATH)

- Create the Tomcat environment setup script (*Tomcat_install_dir/bin/setenv.sh*). This shell script is automatically executed by the Tomcat start-up scripts on UNIX/Linux systems.
- In the file *bin/setenv.sh* enter the following commands:

```
#!/bin/sh
. <AionBRE_install_dir>/aion.sh
```

Note: There is a period in the first position of the 2nd line.

AionBRE_install_dir is the full directory path where CA Aion BRE is installed on the UNIX/Linux platform.

After installing Apache Axis install additional jar files required for Aion BRE web services:

- Copy *AionBRE_install_dir\build\rejava\aiionsession.jar* to *Tomcat_install_dir\webapps\axis\WEB-INF\lib*
- Copy *AionBRE_install_dir\build\rejava\axisaiionsession.jar* to *Tomcat_install_dir\webapps\axis\WEB-INF\lib*

Validate the Apache Axis Setup

After installing Apache Axis 1.4 or higher in the Tomcat \webapps directory, start Tomcat and validate the Axis configuration using the Axis Happiness Page <http://localhost:CA Portal/axis/happiness.jsp>

On the Axis Happiness page, verify the following:

- Check that all of the needed components have been found.
- In the system properties section, locate the Java library path entry (*java.library.path*) and check that the Aion BRE \userlibs directory is defined on this path.

Generate an Aion BRE Component as a Web Service

CA Aion BRE provides a wizard for creating an interface layer that interfaces with the Axis SOAP engine and for registering that interface (exposed methods) with the Axis servlet. Aion BRE components should be programmed following the standards given in the section Programming Aion BRE Components as Web Services.

To generate the Aion BRE application as a Web service, select Tools, Web Service Wizard.

Web services allow only one class to be deployed, selected from among the exported classes of your application. Other exported classes can be used to support complex data types used by the methods of this Web service class. The Web Service Wizard also allows you to select a subset of the public methods of the Web service class for deployment as specific Web services.

The wizard generates a default name for the Web service by which the Web service is identified to clients. The default name of the Web service is *appnameWebService*.

Important! This name must not contain spaces. When a Web service client invokes an Aion BRE Web service, the Microsoft Windows DLLs in the AionBRE \userlibs folder become locked. If the user then modifies the Aion BRE application, the application server must be shut down and restarted before the updated Aion BRE application can be redeployed as a Web service.

More Information:

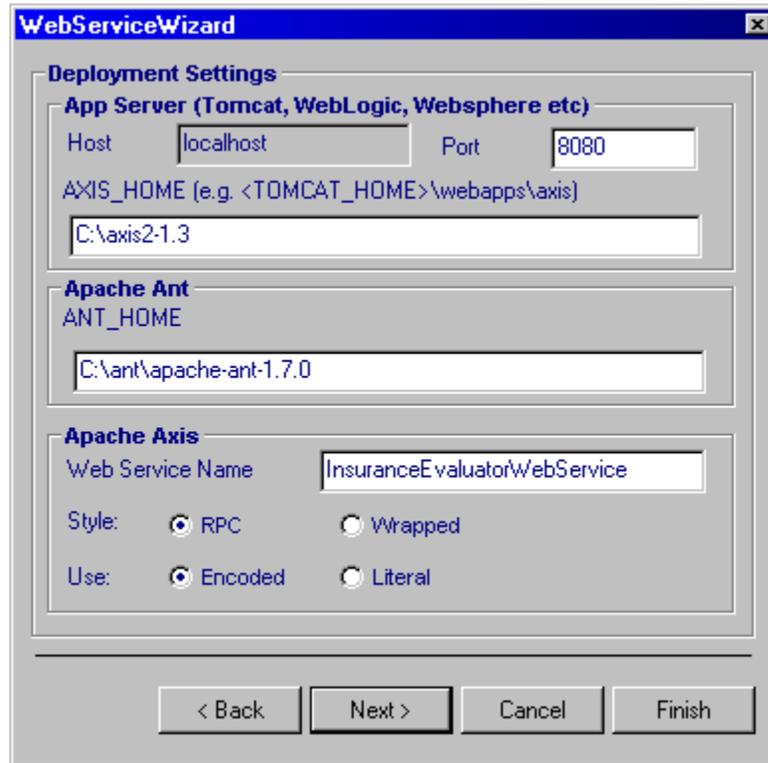
[Use Complex Data Types](#) (see page 400)

Deploy Aion BRE Web Service Components on Microsoft Windows

1. Build your Aion BRE application.
2. Define those public methods for the exported class that will be deployed as a Web service.
3. Export the classes (if any) that define complex datatypes used by the Web service methods.
4. Define public accessor methods for each complex data type class.
5. Select Tools, Web Service Wizard...

The Web Service Wizard dialog appears, showing the list of all exported classes.

6. Select the class to be deployed as a Web service and then click Next.
The methods of the selected class are displayed.
7. Select the methods to be included in the Web service and then click Next.
The Deployment Settings dialog appears.



8. Click Finish.
For more information about Tomcat, ANT, and Axis settings, see <http://ws.apache.org>.

Deploy Aion BRE Web Service Components on UNIX/Linux

1. Build the Aion BRE application following the programming guidelines in the *CA Aion BRE User Guide*.
2. Use the IDE Web Service Wizard to deploy the Aion BRE application on Windows. This generates the Axis WSDD files for the Aion service.
3. Copy the Aion BRE application and generated WSDD files to the UNIX/Linux system.
4. Use the respawn utility on UNIX/Linux system to build the Aion BRE application libraries and Java wrapper jar files.
5. Copy the compiled Aion BRE libraries to the *AionBRE_install_dir/userlibs* directory.
6. Copy the Aion BRE application .jar file to the Axis lib directory.
7. Use the generated WSDD file with the Axis AdminClient to deploy the Aion BRE Web service: `java org.apache.axis.client.AdminClient deploy_AppName.wsdd`

Prerequisite

Obtain Apache Axis 1.4 and higher from <http://ws.apache.org/axis>.

Prepare for WebLogic Deployment

To prepare the system for WebLogic deployment

1. Copy the Apache Axis /webapps directory to a location that WebLogic can refer to. For example:
`Axis_install_dir/webapps/axis`
2. Copy the following files from the directory *AionBRE_install_dir/build/rejava* to the Axis directory WEB-INF/lib:
 - `aionsession.jar`
 - `axisaionsession.jar`

3. Depending on the operating system, perform the following steps:

For Windows:

- Edit the startup script as follows for the WebLogic startExamplesServer.cmd file, which is found in *BEA_install_dir\weblogic81\samples\domains\examples*.
- Append the directory *\axis\WEB-INF\lib* to the CLASSPATH. (near the end of the file)
- Ensure that the system PATH includes *AionBRE_install_dir\userlibs*.

For UNIX/Linux:

- Edit the startup script for WebLogic, startExamplesServer.sh, which is found in *BEA_install_dir/weblogic81/samples/domains/examples*.
- Append the directory */axis/WEB-INF/lib* to the CLASSPATH.
- Ensure that the system library path (LIBPATH, SHLIB_PATH or LD_LIBRARY_PATH) includes *AionBRE_install_dir/userlibs*.

4. Start the WebLogic server by running startExamplesServer.
5. Log into the admin console for WebLogic at <http://localhost:7001/console>.
6. On the left, expand Deployments/Web Application Modules.
7. Choose to deploy a new Web Application Module.
8. Browse to the location of the Axis directory.
9. Enter a name for the module (axis) and deploy.

You should be able to see <http://localhost:7001/axis>, which is the HappyAxis page.

10. To verify that the setup is working, click on the Axis Validate link, look for the java.library.path setting and make sure that *AionBRE_install_dir/userlibs* is included.

Prepare for WebSphere Deployment

To prepare the system for WebSphere deployment

1. Created a .war file that contains the Axis directory contents. For example: *Axis_install_dir/webapps/axis*.
2. Copy the following files from the directory *AionBRE_install_dir/build/rejava* to the Axis directory WEB-INF/lib:
 - aionsession.jar
 - axisaionsession.jar

3. Depending on the operating system, perform the following:

For Windows:

- Edit the startup script as follows for the WebSphere setupCmdLin.bat file, which is found in *WebSphere_install_dir*\Appserver\bin.
- Append the directory *Axis_install_dir*\webapps\WEB-INF\lib to the WAS_CLASSPATH variable.
- Make sure you have the Aion BRE \userlibs directory (*AionBRE_install_dir*\userlibs) referenced in the system PATH.
- Start the WebSphere server by starting the WebSphere Windows service.

For UNIX/Linux:

- Edit the startup script for WebSphere, setupCmdLin.sh which is found in *WebSphere_install_dir*/appserver/bin.
- Append the directory /axis/WEB-INF/lib to the WAS_CLASSPATH variable.
- Ensure that the system library path (LIBPATH, SHLIB_PATH or LD_LIBRARY_PATH) includes *AionBRE_install_dir*/userlibs.
- Start the WebSphere server by running the following:

```
startServer.sh server
```

Where *server* is the server instance (typically "server1").

Log into the admin console for WebSphere at <http://localhost:9090/admin>.

4. On the left, expand Applications, and click Install New Application.
5. Enter the path to the .war file location. For Context Root, enter **axis**.
6. Proceed through the configuration screens, choosing the defaults.
7. In Step 1: AppDeployment Options, modify the following parameters:

Directory to Install Application

Specify a directory as *Axis_install_dir*, where you want to deploy the war file. This directory can then be referenced in the AionBRE ant scripts for deployment of Web service jar files. When the Axis war file is deployed it gets expanded to *Axis_install_dir*/axis.ear/axis.war/WEB-INF/lib.

Application Name

Change the application name from *axis_war* to *axis*.

Enable Class Reloading

Select this option to allow Aion BRE jars to be reloaded when copied to the Axis lib directory.

8. For the .war file to be deployed correctly, you may need to manually start the Axis module using the admin console.
9. You should now be able to see <http://localhost:9080/axis>, which is the the HappyAxis page. To verify that the setup is working, click the Axis Validate link, look for the `java.library.path` setting and ensure that `AionBRE_install_dir/userlibs` is included.

Deploy the Aion BRE Application

The XML files are located in the following directories:

- `AionBRE_install_dir/build/wswizard.xml`
- `AionBRE_install_dir/examples/webservices/PassingObjects/build.xml`
`AionBRE_install_dir/examples/webservices/PassingObjects/aion/build.xml`

To prepare the XML files for deployment

1. Change *port* from 8080 to one of the following:
 - 7001 for WebLogic
 - 9080 for WebSphere
2. For the two build.xml files, changed `axis.home` to refer to the Axis directory.
3. For the wswizard.xml file, change `axislib.dir` to refer to the Axis directory.
4. Run the Ant scripts to deploy the Aion BRE application, as follows:

```
ant deploy-aion
```
5. Verify that the webservice is deployed at `http://localhost:CA Portal/axis/services`.
6. Before running the client, edit the build.xml file found in:
 - `AionBRE_install_dir/examples/webservices/PassingObjects/wsclient/java`
 - On the Windows .NET install, modify the build.xml file found in:
`AionBRE_install_dir/examples/webservices/PassingObjects/wsclient/C#`
7. Change the port from 8080 to one of the following:
 - 7001 for WebLogic
 - 9080 for WebSphere

8. Change `axis.home` to refer to the Axis directory.
9. To build and execute the client, do one of the following:

```
ant run-java-client
```

or

```
ant run-C#-client
```
10. Verify the output from the client.

Code Generation for Web Service Deployment

The Web Service Wizard invokes the CA Aion BRE build process to generate the component with a modified Java interface layer. The wizard then accesses Apache Axis to register the Aion BRE component. The Aion Web services interface employs the Java interface layer as its underlying technology.

Review the current *CA Aion BRE* Readme for any Known Issues regarding support for Web services in CA Aion BRE.

More Information:

[Elements in the Java Interface Layer](#) (see page 364)

Client Programming Considerations

The objective of Web services technology is to enable a client written in any language and running on any platform to call a service Provider written in any language and running any platform. This section covers the requirements of Web service client programming in only the broadest terms.

Building a client involves one of the following alternatives:

- For a Java client: download the appropriate Apache tools (including WSDL2JAVA for translating WSDL specifications of a Web services into Java code) and SOAP libraries for inclusion into the Java program.
- For a C or C++ client: obtain the Microsoft SOAP Toolkit 2.0.
- For a Managed C++, VB.Net, or C# client, you must have Visual Studio .NET installed. You need the .NET framework, .NET SDK, and Visual C# or Visual Basic.Net development environment portions of that install. You must have also installed CA Aion BRE, specifying Visual Studio.NET as the compiler when you set it up. This may sound complicated, but the programming of a Web services client under .NET is an order of magnitude easier than programming such a client in Java.

Note: In general, Aion BRE components cannot call Web services. However, an Aion BRE component can invoke a Rule Manager rulebase that has been deployed as a Web service.

The CA Aion BRE example set provides several examples of Aion BRE applications that can be deployed as Web services along with code for Java clients that invoke these services. To compile the Java programs you will need the proper Apache libraries. ANT scripts are provided for compiling these programs. ANT, a utility for automating a system build process, is freely downloadable from:

<http://jakarta.apache.org/ant/index.html>

To run the ANT scripts:

- Define a system variable ANT_HOME pointing to your ANT directory.
- Put %ANT_HOME%\bin on your system path.

You should have a system variable JAVA_HOME pointing to your Java SDK directory.

ANT scripts may also be used to compile the Aion BRE component.

For further information on ANT, see the readme.txt file in the \examples\java folder.

Administering Aion-Based Web services

Administration of Web services is accomplished using Apache Axis. You can access the administrative function at this address:

<http://localhost:8080/Axis>

Additional Resources on Web Services

As the primary resource for Aion BRE Web services deployed under Apache Axis, visit:

- <http://ws.apache.org/axis>

For more information on Web services, visit:

- <http://www.oasis-open.org/> and <http://www.w3c.org>

Web services rely heavily on the eXtensible Markup Language (XML) which has emerged as the universal language for data communication. For more information on XML, you might begin by visiting the following sites:

- <http://www.w3c.org/xml> (the W3C owns the XML standard)
- <http://www.ucc.ie/xml/#index> (which provides an XML FAQ)

For more information on .NET, visit:

- <http://www.microsoft.com> and follow links to .NET

For more information on IBM, visit:

- <http://www-01.ibm.com/software/solutions/soa/>
- <http://alphaworks.ibm.com/webservices>

or more information on Open Source Apache Foundation visit:

- <http://www.apache.org>

Chapter 18: Debug Aion BRE Applications

Use the Aion BRE Debugger to locate problems within your Aion BRE application. By running an Aion BRE application in debug mode, you can view its execution flow, the values of variables, and other troubleshooting information.

The Aion Debugger is an application that is, itself, written using Aion. When running an Aion BRE application in Debugger mode, the application runs normally—windows and dialog boxes display and accept user input. However, through the Debugger you can also:

- See the body of the currently executing method.
- Step through the application method-by-method or line-by-line. Method logic executes as you step through the methods.
- Set breakpoints to strategically suspend execution. You can set breakpoints at the start of a method, at any executable line within a method (rule methods also), or when a specified attribute is changed.
- View and modify the values of variables as you step through a method.
- View the method call stack that has been traversed prior to arriving at the current point in the execution.
- Build components using an embedded interpreter to enable debugging of deployed components (from client applications); see the Embedded Component Debugging section.

You can debug any Aion BRE application whose *entry class* includes a *Start* method. When you start the Debugger, Aion executes an interpreted version of the application, which is run without explicit compiling.

When debugging an application, the Debugger runs the application as usual, while also providing facilities for viewing and controlling source code and data during execution. You can see both what is happening and how it happens.

What you see while debugging an application depends on how you step through the application and where you set your breakpoints and watchpoints. Debugging can include the following general steps (each step is discussed further in the following sections of this chapter):

1. From the Aion BRE application, start the Debugger.
2. Set code breakpoints at executable lines in the code or at the start of methods.
3. Set watchpoints to identify attributes whose values you want to monitor.
4. Set data breakpoints for specified attributes so that Debugger will stop at any line of code that changes the attribute's value.
5. Start program execution either by running to the next breakpoint or by stepping in, over, and out of the code line-by-line or method-by-method.
6. View (or modify) attribute values throughout execution, and watch the flow of execution.
7. If necessary, move, remove, or add breakpoints and watchpoints, and reexecute the application.

This section contains the following topics:

[Debugger Features](#) (see page 415)

[The Debugger Window](#) (see page 418)

[Breakpoints](#) (see page 429)

[Watchpoints](#) (see page 432)

[Debugger Settings](#) (see page 432)

[Debug Aion BRE Applications](#) (see page 433)

[Debugging Rule-Based Inference](#) (see page 436)

[Shut Down the Debug Session](#) (see page 438)

Debugger Features

The following list summarizes the tasks you can perform with the Debugger. You can access most of these actions from the menu, toolbar, right-click shortcut menu, or using keyboard accelerators:

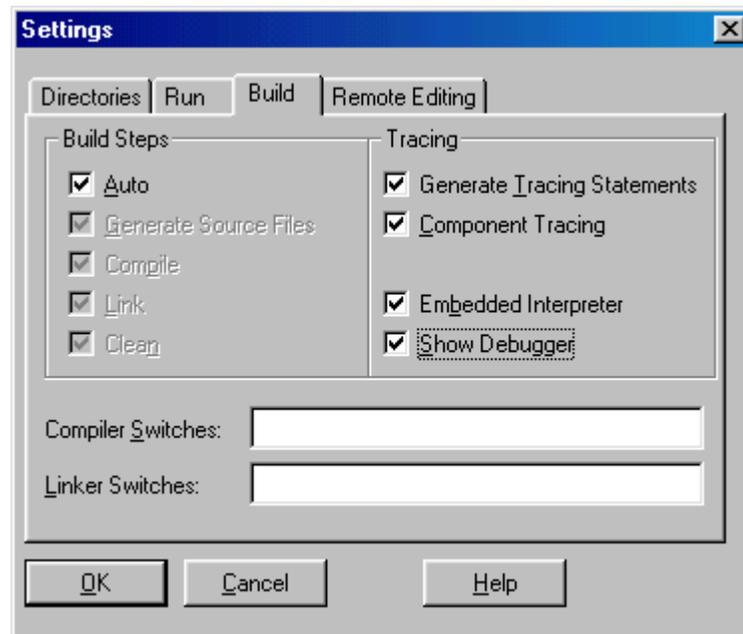
- Navigate through method code, including knowledge methods (rules).
- Configure and generate a trace file to selectively track the Debug session.
- Set and clear data watchpoints.
- Set and clear data breakpoints.
- Dynamically change data values (including to null or unknown)
- Set and clear method breakpoints.
- Manually step into, over, and out of methods during execution.
- Run or continue execution to the next breakpoint (data or code), or to the end of the application, whichever is first.

Embedded Component Debugging

Debugging of embedded, or “wrapped”, components presents special challenges. Ideally, you want to preserve the original operating environment of the component, that is, allow the wrapper program to be run compiled while still providing full debugging facilities for the component. To meet this challenge, Aion provides Build options for generating debugging information from an embedded component.

Note: Embedded component debugging is available only for components developed under Microsoft Windows.

Aion provides three options for building components on the File, Settings Build tab. These options address different levels for obtaining debugging information from built Aion BRE applications.



- Component Tracing:** Obtains a production trace from a compiled component. A compiled trace is not as detailed as the Debugger trace (for example, data assignments are not included). This option provides the typical compiled trace for embedded components. To activate this option, you must also activate the "Generate Tracing Statements". The trace is generated in the Aion specified directory for the Build trace.

- **Embedded Debugger:** Obtains the Aion debugger trace from the component.

This option causes the Aion interpreter to be called internally when the component executes. It is possible to generate a full trace, including data assignments. Running a component under this option, however, means that the component is much larger than if it were built without this option, and it runs much slower. Aion provides a warning messages at build-time regarding module size and performance when building under this option.

The Embedded Interpreter is subject to two restrictions:

- The application file and its \bin directory must be in same location (subdirectory).
- Because only one interpreter can run within a Window thread, a component built with the embedded interpreter can be run from an Aion client (wrapper) in compiled mode but not in interpretative mode.

- **Show Debugger:** Runs the Aion interactive debugger itself on the component.

This is the most powerful feature of Embedded Component Debugging: the ability to operate within the Aion debugger when running a built Aion component. Selecting this option allows you to step through the debugger as if you were running the Aion code interpretively within the Aion development environment. The Aion debugger window appears as soon as the component begins to execute.

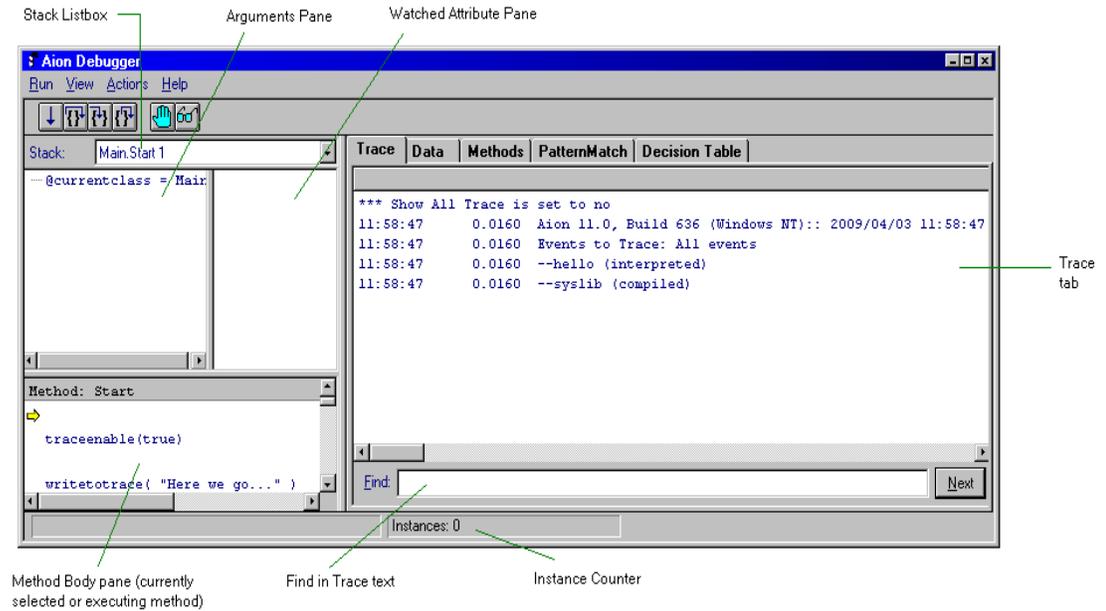
This option requires that the Embedded Interpreter is turned ON, and is subject to the same restrictions as the Embedded Interpreter.

The Debugger Window

To start debugging an application: Click the Debug button on the Aion toolbar (or from the File menu, choose Run, Debug).

The Debugger starts, displaying the body for the Start method and the current Trace information:

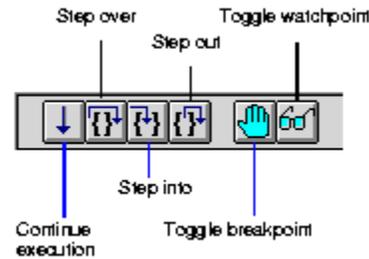
Equation 1: Shows the debugger window



The Debugger contains resizable panes separated by splitters. Each tab contains additional panes. You can re-size the panes to see all of the Debugger fields.

The Debugger Toolbar

You can access debugger-related functions from the Debugger's menu or its toolbar. Following is the toolbar for the Debugger:



For additional information on the toolbar buttons, see their corresponding sections:



Continue execution-Run the application to the next breakpoint or if none exist, to end of the application. See Controlling the Flow of Execution.



Step over call to next line in this method-Execute the next line in the current method, and if it is a method call, execute the entire method without stepping into it. See Controlling the Flow of Execution.



Step into next executable line-Execute only the next consecutive line in the current method and pause the Debugger. See Controlling the Flow of Execution.



Step out of this method-Finish executing the current method or line and return to the calling method or line. See Controlling the Flow of Execution.



Toggle breakpoint-Set or remove a breakpoint for a line of code or a rule name. See Breakpoints.



Toggle watchpoint-Set or remove a watchpoint for an attribute whose value you want to monitor. See Watchpoints.

Stack List Box

The following figure shows a Stack list box:



The Stack list box displays the Call Stack, which consists of all method calls that have not returned-in other words, a nested list of suspended methods. The Debugger automatically updates the Stack list box and displays the current method in the form *class.method*. If the method is an instance method, the “current” instance is included in the list box as the first entry. This simplifies accessing the attribute values for the current instance. During inferencing, the Stack list box displays inferencing information instead of *class.method*.

The Method Body pane displays the code of the method currently selected in the Stack list box. By default, the currently executing method is selected in the Stack list box.

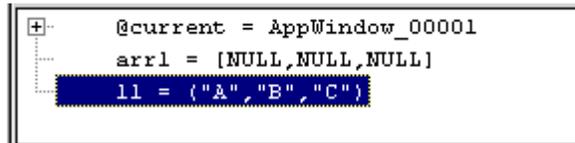
You can open the Stack list box to view or navigate through the list of methods and their corresponding method bodies. When you select a different method in the Call Stack, the Debugger displays the method's code into the Method Body pane so you can examine the flow of the application. Doing this, however, does not cause the position of the execution arrow to change. Regardless of which method is displayed in the Method Body pane, actual execution continues from the line indicated by the execution arrow on the *currently* executing method. The currently executing method is the "top" method in the stack.

More Information:

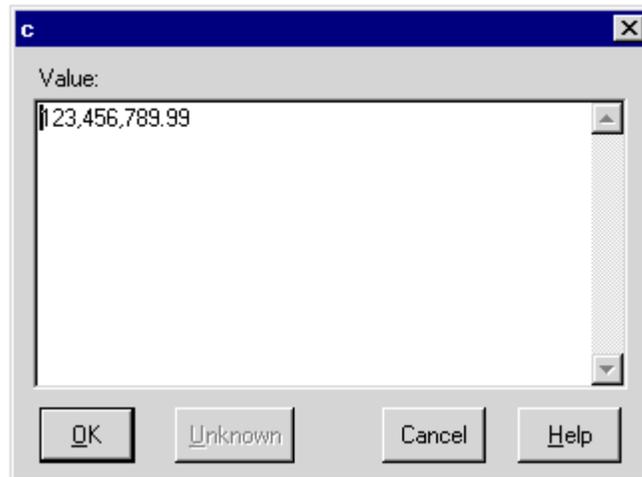
[Use the Call Stack](#) (see page 435)

Arguments Pane

The Arguments pane, as shown in the following figure, displays the values for all input, output, and local variables in the currently executing method:



You can click the plus symbol to expand an attribute, and double-click on any item to display (or modify) the current value.



For information about changing attribute values during debugging, see [Viewing and Modifying Data Values](#) in the CA Aion BRE online help.

Watched Attribute Pane

This pane displays the values of any attributes and instances for which watchpoints or data breakpoints have been set. You specify which class and instance attribute values you want to explicitly watch by setting data breakpoints and watchpoints on the Data tab.

For more information about setting data breakpoints, see [Breakpoints](#) in the CA Aion BRE online help.

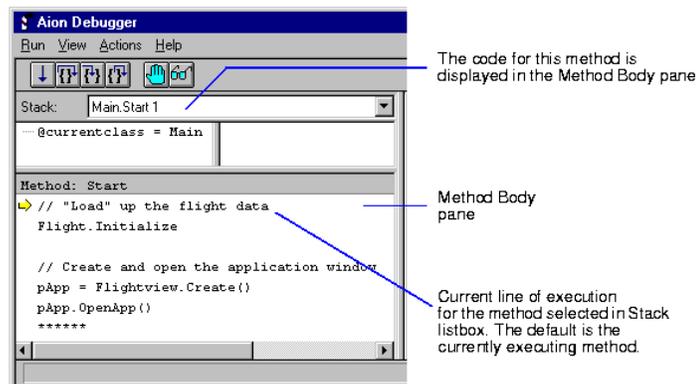
More Information:

[Watchpoints](#) (see page 432)

Method Body Pane

Below the Arguments and Watched Attributes panes is the Method Body pane. This pane contains the body of the method displayed in the Stack field. Typically, this is the currently executing method.

A yellow arrow marks the current line of execution for the executing method or, if you are using the Stack list box to navigate through suspended methods, the current line of the selected suspended method. When the Debugger first starts, the arrow points at the first line of the Start method, as shown in the following figure:



Watch this area as you step through the code. You can set and undo breakpoints from the Method Body pane or from the Methods tab.

To adjust the sizes of the Method, Arguments, and Watched Attribute panes, position the mouse pointer on the border of the pane you want to resize, then left click and drag to re-size the pane.

Note: To display methods that are not in the call stack, use the Methods tab.

Instance Counter

The Instance counter shows the current number of dynamic instances in the application being debugged. Use the Instance counter to help detect when the application is stranding instances.

Note: Static instances are not included in the Instance Counter.

Debugger Tab Pages

The Debugger includes five tabbed pages, each containing a different type of information about the application being debugged:



Trace

Displays the execution history for the debug session.

Data

Lists the included libraries, their classes and instances, and provides windows for viewing class and instance attributes and specifying watchpoints and data breakpoints.

Methods

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

PatternMatch

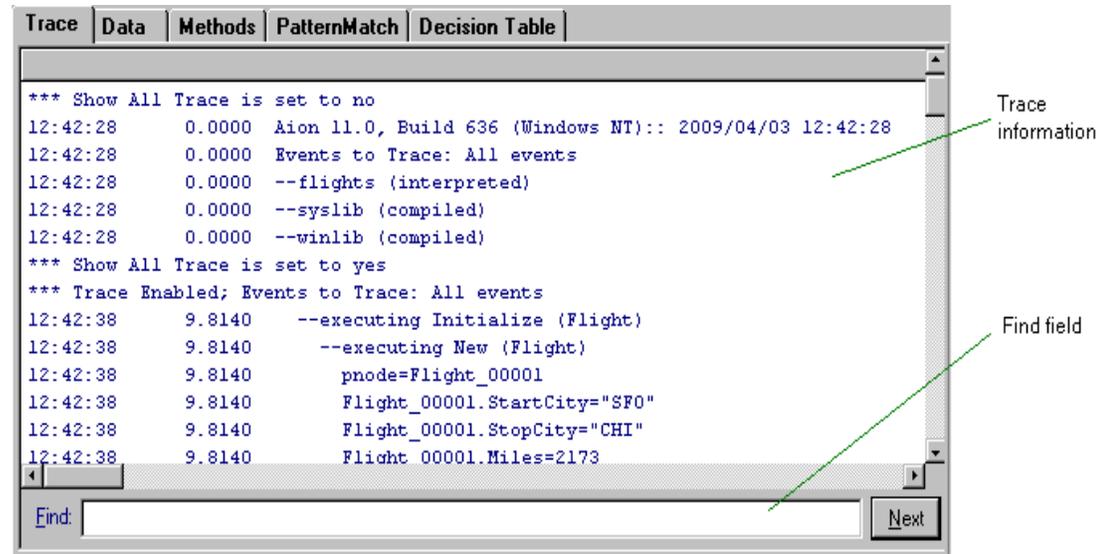
Displays the Binding and Pattern Matching values for executed IFMATCH rules (forward-chaining only).

Decision Table

Displays a graphical view of an active Decision Table.

Trace Page

The Trace page displays the execution trace of the debugged application:



You can use the Debugger Settings dialog to configure what displays on the Trace page (see Debugger Settings). The default information displayed, a subset of what Aion writes to the Aion trace (.trc) file, includes a list of all libraries referenced by the application and whether they are opened in interpreted or compiled mode. Optionally, the Trace page can list each method as it executes, the value assignments that occur during method execution, and database communications.

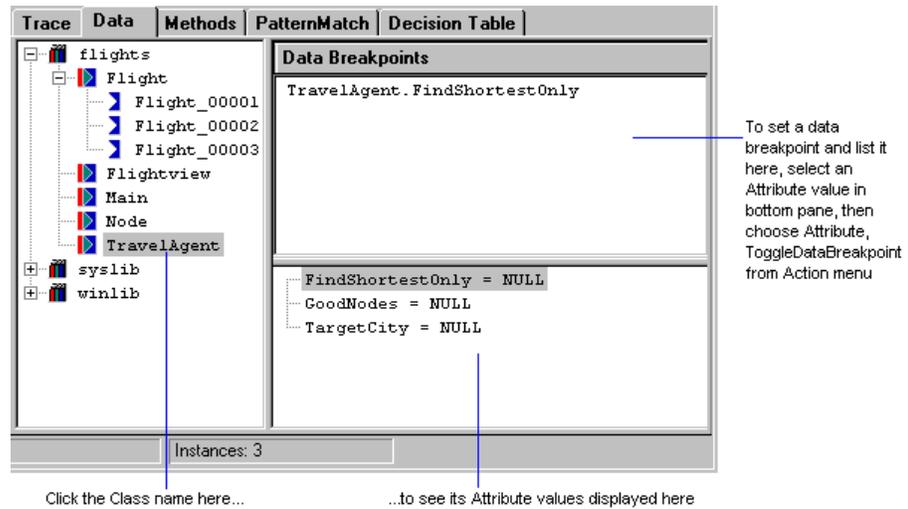
- Trace output can also include the following inference information:
- Begin/end of INFER block
- Rules posted, pending, failed, and fired
- Rule premise evaluated
- Rule actions started and completed
- TM retraction and confirmation
- Begin/end Backwardchain
- Begin/end ForwardChain
- Bound instances for the start of a pattern match rule action

Use the Trace page's Find field to search the body of the Trace window for a specific word, letter, or number.

Data Page

You use the Data page to display classes and instances, their attributes and associated values, and to set data watchpoints and data breakpoints for attributes so you can track their values as the application executes.

Initially, the Data page displays a Data tree consisting of an alphabetical list of the application's libraries. You can expand the tree to see a library's class by clicking the + symbol next to a library. Click the + symbol next to a class to see its instances.



When you highlight a class in the Data tree, the lower-right pane displays the affiliated class attributes. Attributes are added to the list as they are initialized, and their values can change as the program runs.

When you highlight an instance in the Data tree, the pane displays affiliated instance attributes.

The top-right pane displays all data breakpoints that you set.

Note: The Data page does not show constants.

When you set a data watchpoint for an attribute, Aion adds the attribute to the Watched Attribute pane (in the top-right corner of the Debugger main window). You can watch the attribute's value change as you step through your application.

When you set a Data Breakpoint on an attribute, Aion adds the attribute to both the Data Breakpoints pane (top-right pane of the Data page) and the Watched Attribute pane. .

While stepping through the application, you can dynamically change attribute values. Double-click on the attribute displayed on the Data page or, if it is a watched attribute, from the Watched Attribute pane, and change the value in the pop-up window.

More Information:

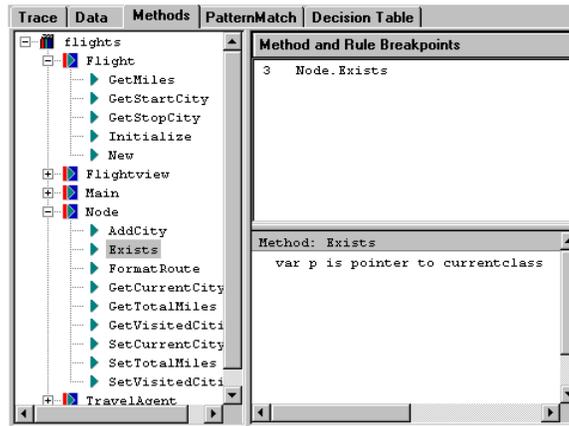
[Set and Remove Watchpoints](#) (see page 432)

[Set and Remove Data Breakpoints](#) (see page 430)

[View and Modify Data Values](#) (see page 434)

Methods Page

Use the Methods page to set and clear code breakpoints in the application, to see which breakpoints have been set, and to display the code for any method in the application.



A breakpoint is indicated by a red dot in a method's body. It specifies that execution will stop prior to executing this line.

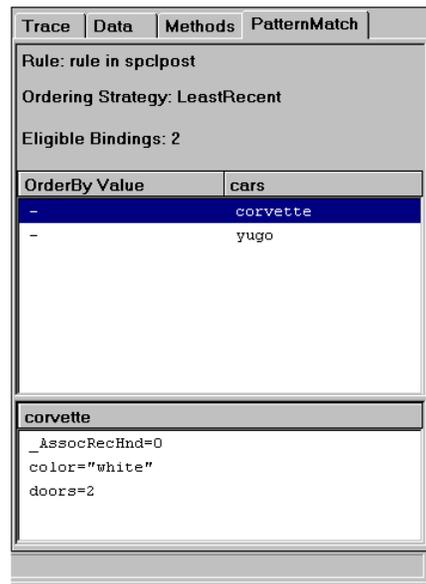
More Information:

[Breakpoints](#) (see page 429)

[Set and Remove Code Breakpoints](#) (see page 431)

PatternMatch Page

The PatternMatch page displays bindings and instances for IFMATCH pattern-matching rules that the Debugger encounters during execution. Values display only during forward chaining.



When the Debugger encounters an IFMATCH condition, this page lists the rule name, the number of eligible bindings (excluding the current bind), the pattern match instances, as well as the binding ordering strategy (LeastRecent or MostRecent).

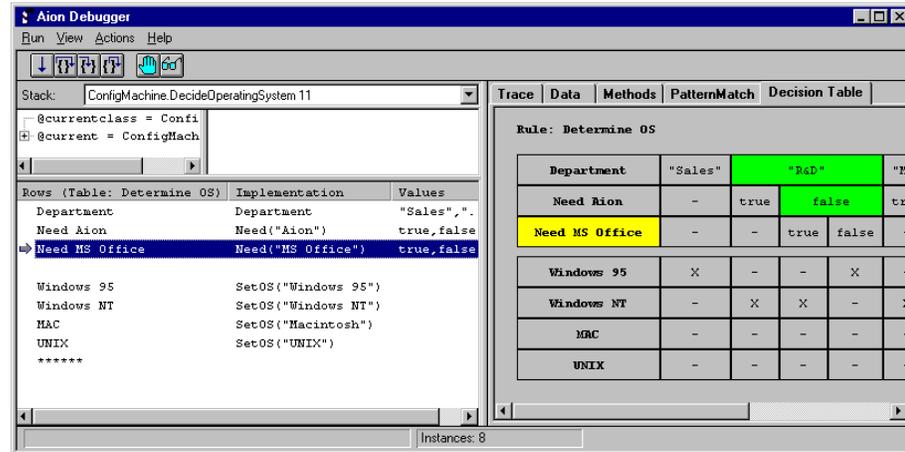
Note: This page does not display the current binding on which a rule is currently executing. The current binding displays in the Method Body pane during the WHERE clause evaluation and the execution of the rule action.

More Information:

[Debugging Rule-Based Inference](#) (see page 436)

Decision Table Page

During debug mode, when a Decision Table becomes active (posted, evaluated, or executed), the Decision Table rule displays in the Method Body pane, while at the same time the Decision Table page presents a graphical view of the active Decision Table.



The Decision Table rule displays in the Debugger's Method Body pane in a format different from method code. The display format consists of separate rows for each Condition, followed by a blank line, and then separate rows for each Action. This is also how the Decision Table displays in the Methods tab when setting breakpoints.

For all condition and action rows:

- Column one contains the condition or action name. Its title area includes the rule name.
- Column two contains the implementation.
- For conditions, the test values display in column three.

You can resize the columns by dragging the column title separators.

In addition, you can:

- Step into rows (conditions and actions) of a Decision Table (F11).
Stepping into a condition or action steps into the method's implementation.
- Step over rows (conditions and actions) in a Decision Table (F10).

As you step over the conditions of a decision table, several things happen:

- The current row to be evaluated is indicated in the Method Body pane with a right arrow.
- The current row to be executed is highlighted in the graphical view on the Decision Table page.
- In the Arguments pane, evaluated conditions are assigned an internally-created variable named `_condition_n` (where *n* is a variable beginning at 1). The result of each evaluated condition is stored in the defined variables.
- Evaluated conditions with matching test values cause the appropriate cell in the Decision Table page to be highlighted. The highlighting persists for the duration of the Decision Table evaluation.

Breakpoints

Breakpoints let you strategically suspend execution of an application running in debug mode. You can assign a breakpoint to a method, to any executable line within a method, or you can set data breakpoints for specified attributes so that Debugger will stop at any line of code that changes the attributes value.

When the application encounters a breakpoint, the Debugger comes to the foreground and the method containing the breakpoint displays. From this point, the Debugger lets you step through method logic, line by line if necessary, analyzing the effects each step has on the application.

There are two basic types of breakpoints: data and code.

Data Breakpoints

Data Breakpoints are marked attributes that indicate a point at which the Debugger should stop before allowing the runtime system to continue to execute. The Debugger will stop prior to any line of code about to execute if that line of code changes the marked attribute's value.

One data breakpoint could potentially cause the Debugger to stop at many different places during execution.

Code Breakpoints

Code Breakpoints are marked lines of code that indicate a point at which the Debugger should stop before allowing the runtime system to continue to execute. Code breakpoints consist of two sub-types: method and rule.

- Method breakpoints mark lines of code at which the Debugger should stop before proceeding with execution. This can be any executable line of code in your interpretive application.
- Rule breakpoints are set on rule names. They cause the Debugger to stop when Aion initially posts one of these rules to the inference engine and again when the engine evaluates the rule's premise.

Set and Remove Data Breakpoints

For step-by-step procedures for setting and removing data breakpoints, see [Setting and Removing Data Breakpoints](#) in the CA Aion BREonline help.

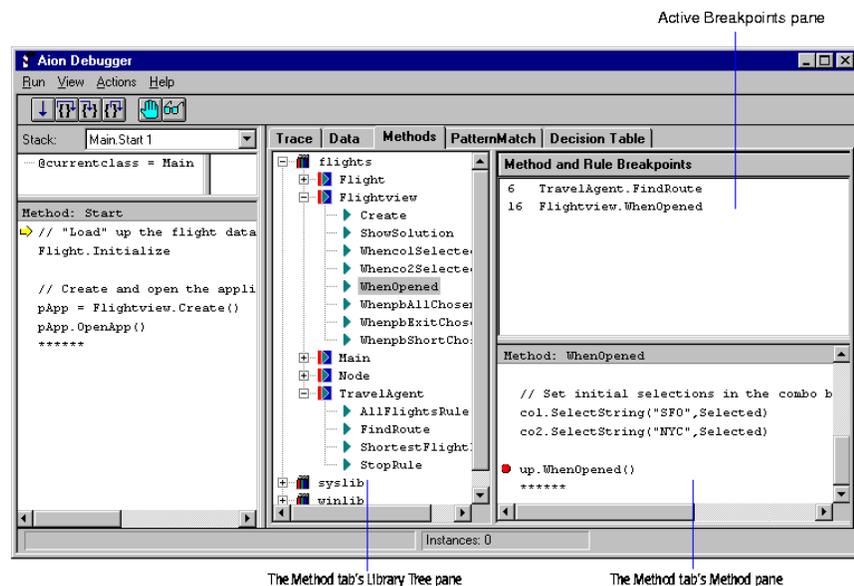
Set and Remove Code Breakpoints

You can set an unlimited number of code breakpoints for an application. Though code breakpoints persist from session to session, they apply only during debug sessions.

You can assign a breakpoint at the beginning of a method, at an executable line *within* a method body, or on a rule name.

You cannot set breakpoints in or step into compiled libraries (such as the primary libraries included with Aion BRE).

If you set a code breakpoint on a non-executable line (such as a comment, blank line, or local variable definition), the Debugger will not stop at that breakpoint during execution, unless the non-executable line is the first line of a method.



Active Method and Rule Breakpoints Pane

Every code breakpoint you set displays in the Active Method and Rule Breakpoints pane (in the upper-right area of the Methods tab).

Breakpoints display either in the form `n class.method` where `n` is the line-number in method, or in the form `rule name in method`.

Set Breakpoints from the Methods Tab

You can set code breakpoints from either the Methods tab or in the Debugger's Method Body pane.

See Setting Breakpoints from the Methods Tab in the CA Aion BRE online help for step-by-step procedures relating to the following topics:

- Setting breakpoints at the beginning of a method
- Setting a breakpoint at an executable line within the method body
- Setting a breakpoint on a rule name
- Removing a code breakpoint

Watchpoints

A watchpoint lets you specify an attribute whose value you want to monitor. When you set a watchpoint, during execution the value for the corresponding attribute displays in the Watchpoint pane above the Method Body pane.

Note: You can set watchpoints only for instance or class attributes. You cannot set watchpoints for local variables, input arguments, or output arguments. These are already visible in the Arguments pane.

Set and Remove Watchpoints

For step-by-step procedures for setting and removing data watchpoints, see Setting and Removing Data Watchpoints in the CA Aion BRE online help.

Debugger Settings

Use the Debugger's Settings dialog to customize how information displays for the Debugger. Modifications to the Split Style, Show Tool Tips and Show Status Bar settings take effect when you restart the Debugger. All other modifications to the Settings dialog take effect upon close of the dialog. Aion saves the settings from session to session.

Configure Debugger Settings

For step-by-step procedures for changing the debugger's settings, see Configuring Debugger Settings in the CA Aion BRE online help.

Debug Aion BRE Applications

This section describes the basic steps in the debugging process.

To start the Aion Debugger, from the Aion BRE application window, click the Debug button. Aion BRE opens the Debugger, starts the application in debug mode, pausing execution at the first line of the Start method body.

Set Breakpoints and Watchpoints

See Breakpoints for a description of breakpoints. See Watchpoints for a description of watchpoints.

For step-by-step procedures for setting and removing data breakpoints and watchpoints, see the following topics in the CA Aion BRE online help:

- Setting and Removing Data Breakpoints
- Setting and Removing Watchpoints

Control the Flow of Execution

The Debugger can execute from breakpoint to breakpoint, line-by-line, or method-to-method. Depending on how you want to view the code, you can use one or more of these options. At each stopping point in the execution, you choose whether you want to continue execution to the next breakpoint or step through the code.

To run to the next breakpoint, do one of the following:

- Click the Continue execution toolbar button.
Or
- Press F5.
Or
- From the Run menu, choose Go.

If no breakpoints are set beyond the current line of execution, the Debugger runs to the end of the application.

To control the flow of execution, do one of the following:

- Use the buttons for Step In, Step Out, and Step Over.
Or
- Use the corresponding options on the Debugger Run menu or the shortcut keys indicated next to each menu option.

Step In



When you choose Step In, the following occurs:

- The next consecutive line in the method is executed.
- If the line calls a method, it becomes the current method and displays in the Method Body pane window.
- The yellow arrow is positioned at the first line of that method.

Step Out



When you choose Step Out, the Debugger exits the current method or line, and returns to the calling method or line.

Step Over



If you choose Step Over, the Debugger executes the next line in the current method. If the line is a method call, the entire method is executed without being stepped into.

Note: If the current method contains a breakpoint, Aion cannot complete execution of the entire method when Step Over is chosen. Execution suspends at the breakpoint line, and the current method displays in the Debugger's Method Body pane.

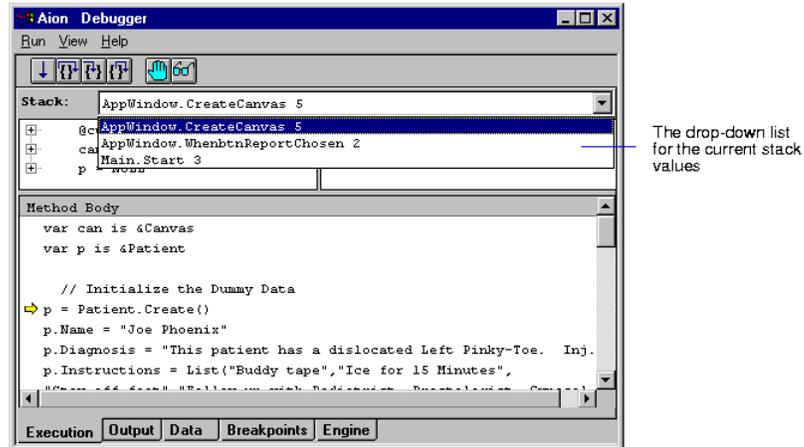
View and Modify Data Values

You can view and modify the values for arguments, local variables and attributes whenever the application suspends execution. The Debugger displays method arguments and local variables in the Arguments pane. Class and Instance attributes display in the Data tab.

For step-by-step procedures for modifying a value for a method argument, local variable, or attribute, see [Viewing and Modifying Data Values](#) in the CA Aion BRE online help.

Use the Call Stack

The call stack—a list of all method calls that have not returned—displays in the Stack list box at the top of the Debugger window:

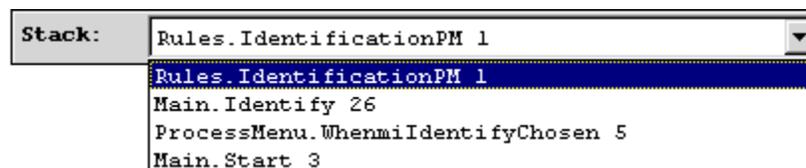


When one method calls another method, the calling method execution is suspended until the called method finishes executing. Consequently, *nested methods* form a chain of active methods—methods whose execution has started but not yet finished. Aion places the names of nested methods on the call stack.

Using the Stack list box on the Debugger window, you can view the call stack to:

- Examine the chain of methods that brought the application to its current stopping point or breakpoint.
- View the body of any method in the chain so you can investigate individual method lines that might have affected the current state of the application—for example, other methods that might have been called or variables that might have been set.

The Call Stack is structured as follows:



- Methods in the chain are organized from most recently called (at the top of the list) to least recently called.
- The method at the bottom of the list is always the Start method of the entry class.
- If a method is an instance method, the *current* instance is included in the call stack as the first entry. This simplifies accessing the attribute values for the current instance.
- The number at the end of each entry specifies the following:
 - For the most recently called method, the current line (the *next* line to be executed in the method).
 - For all other methods, the *last* line executed (the line that called the next higher method listed on the call stack).
 - For Decision Tables, this line number currently has no significance.

To view a method in the call stack

- Click a line in the Stack list box's drop-down list.
Or
- From the Debugger View menu, choose Execution, Previous stack frame, or Next stack frame.

The Debugger displays any arguments and local variables in the upper-left pane, and the method text displays in the lower Method Body pane. As you select different methods in the Call Stack, Aion displays their code in the Debugger panes.

Note: Simply changing which method displays in the panes of the Debugger window does not change which method is “current” (the method containing the next line to be executed). Regardless of which method is displayed in the Method Body pane, execution will continue from the yellow arrow in the top-most method listed in the stack.

Debugging Rule-Based Inference

During rule-based reasoning, the Debugger displays information related to the actions of the inference engine as it processes rules.

Note: For background information on inference engine processing and rule-based reasoning, see the *CA Aion BRE Rules Guide* and CA Aion BRE Rules online help.

Backward Chaining

During backward chaining, the Debugger's Method Body pane displays information about the backward chaining goal stack and about the rules that the inference engine has tried to apply.

For step-by-step procedures for following a backward chaining process in the Debugger, see Backward Chaining in the CA Aion BRE online help.

Forward Chaining

When forward chaining with pattern-matching (IFMATCH) rules, you can display current bindings and the rule-instance bindings for rules in the Debugger as they execute.

For step-by-step procedures for following a forward chaining process in the Debugger, see Forward Chaining in the CA Aion BRE online help.

Special Considerations for Decision Tables

As previously described in the topic, Decision Table Tab in the Debugger Tab Pages section, when you step into a Decision Table at rule-post, evaluation, or execution time in debug mode, the Decision Table tab displays a graphical view of a Decision Table:

Trace	Data	Methods	PatternMatch	Decision Table
Rule: Determine OS				
Department	"Sales"	"R&D"	"Marketing"	"System Admin"
Need Rion	-	true	false	true false true false
Need MS Office	-	-	true false	- - true false -
Windows 95	X	-	- X	- - - - -
Windows NT	-	X	X	- X - X - -
MAC	-	-	-	- - X - - -
UNIX	-	-	-	- - - - X X

Conditions for Debugging a Decision Table

As you step over the conditions of a decision table (F10), several things happen:

- The current row to be evaluated is indicated in the Method Body pane with a right arrow.
If F10 does not advance the current row arrow, it may be because the engine is actually operating on some portion of the implementation, however the current line number in the Stack window should change.
- The current row to be executed is highlighted on the graphical view on the Decision Table tab.
- In the Arguments pane, conditions that are evaluated are assigned an internally created variable named `_condition_n` (where `n` is a variable beginning at 1). The result of each evaluated condition is stored in the defined variables.
- Evaluated conditions with matching test values cause the appropriate cell in the decision table tab to be highlighted. The highlight persists for the duration of the decision table evaluation.

Actions for Debugging a Decision Table

If Aion cannot determine enough information from the conditions to proceed to execute the actions, the Decision Table is pended. When a Decision Table is pended, the last match value will not be shown in the graphical display before the Debugger returns to the Rule List.

If the inferencing process later resets the Decision Table to ready status and re-entered it, none of the previous cell statuses will be shown. The rule will be completely re-evaluated.

Shut Down the Debug Session

To close the Debugger

- Choose Stop Debugging from the Run menu
Or
- Click the X symbol in the upper-right corner of the debug window.

When you close the Debug Window, CA Aion BRE retains all set code breakpoints for use in future debugging sessions.

Chapter 19: Run and Build Applications

This chapter describes how to build a compiled version of an Aion BRE application for deployment in a production environment.

Before an Aion BRE application can be run stand-alone, it must first be built. The build process entails compiling and linking the Aion BRE application using the Microsoft Visual C/C++ compiler. Before building, you can optionally specify additional build directives from within Aion.

This section contains the following topics:

[Run Aion BRE Applications](#) (see page 439)

[Prepare to Build Aion BRE Applications](#) (see page 441)

[Build the Application](#) (see page 443)

[Use Interface Layers](#) (see page 444)

[Deploy the Application](#) (see page 446)

Run Aion BRE Applications

You can run an Aion BRE application in one of two ways:

- Using the Aion interpreted mode (no compiling necessary)
- As a stand-alone executable (must be compiled)

Run Aion BRE Applications Interpretively

The Aion BRE *interpreter* is primarily used for executing an application during its development. If you run a non-compiled Aion BRE application from within Aion, the application is automatically executed in Interpreted mode. Interpreted execution speeds up application development because no compile or link steps are required between editing sessions, and because extensive facilities for runtime tracing and graphical debugging are provided.

Run Stand-Alone Applications

Stand-alone applications run considerably faster than applications running in interpretive mode. Typically, an application is *built* when the development phase is complete, and the stand-alone executable is ready for deployment into a production environment (although it can be built at any time during development as well). Stand-alone applications require the executable files produced for the application and any included libraries.

Compile Applications

To build a stand-alone executable file, you must compile and link your Aion BRE application using the Microsoft Visual C/C++ compiler (version 6.0 or higher). The location of the compiler is the only required compiler-specific configuration information.

Building Applications

The Build process creates a dynamic link library (DLL) that contains a compiled version of all objects defined within the application library. If the application library is designed to be an executable (for example, if the Entry class includes the Start method), the build process also produces an executable file (EXE) that contains the necessary code for initializing the Aion runtime environment.

During the build process, Aion BRE generates code for each method body, invokes the specified compiler to compile the generated code, and then invokes the linker to produce the executable library file.

Filenames

By default, Aion generates filenames of the form appname.dll and _appname.exe (where appname is the name of your application).

For example, if your application is called PUZZLE.APP, building the application produces a DLL named PUZZLE.DLL which contains a compiled version of all the methods defined in the application. An executable file would be named _PUZZLE.EXE.

Note: You can use the Library Properties dialog to assign a custom name to the .EXE and .DLL files.

More Information:

[Prepare to Build Aion BRE Applications](#) (see page 441)

Included Libraries

Any custom libraries that are included by the application library must also be built. However, building them separately is not required if your Build settings are set up appropriately. If you want an application's included libraries to be built automatically at build time, make sure you check the Auto flag on the Build tab of the Settings dialog (choose Settings from the File menu). When this global flag is checked, if a non-built or out-of-date library is included in the application at build-time, the Aion recursive build feature will automatically build the included library as well. See To specify build settings in the Configuring Build Settings section in the CA Aion BRE online help for detailed instructions.

By default, all Aion BRE applications include at least one (and usually more) of the Aion BRE system libraries (such as SYSLib, or WINLib). All Aion BRE system libraries have been pre-built, and so do not require explicit building. Once a library has been built into a DLL, it does not need to be rebuilt unless the library itself is modified, or any library that it includes is modified.

Note: You cannot use the Aion BRE build process to create .DLL files for external methods. You can compile and link external methods only with your C compiler.

Prepare to Build Aion BRE Applications

Before building an application, check the items on the following list:

- Verify that the Build Settings are correct.
Build Settings reflect general information needed for building any Aion BRE application.
- Specify any library-specific build directives.
Library-specific build directives are optional for an application.

Configure Build Settings

Before building an application, use the Build Settings dialog to provide Aion with system-related build information. Aion BRE uses global Build Settings. Once they are set, they apply to all Aion BRE applications; however, the settings can be modified at any time, from any application.

Note: Typically, after you specify the build settings, they will not need to be modified.

For step-by-step procedures for specifying build settings, see Configuring Build Settings in the CA Aion BRE online help.

Directories Tab

For step-by-step procedures for specifying system locations for accessed and generated files, see Directories Tab in the CA Aion BRE online help.

Run Tab

For step-by-step procedures for choosing options for running the built application, see Run Tab in the CA Aion BRE online help.

Build Tab

For step-by-step procedures for choosing which steps to include in the build, see Build Tab in the CA Aion BRE online help.

Configure Library Properties

For each application, you can use the Library Properties dialog to specify build directives and customize path information. Settings in the Library Properties dialog apply only to the application from which the dialog was invoked.

Invoke the Library Properties Dialog

For step-by-step procedures for invoking the library properties dialog, see Invoking the Library Properties Dialog in the CA Aion BRE online help.

Specify the Executable Directory

During the build process for an application, Aion generates a DLL for the application and, optionally, an EXE file. By default, these files are created in the *appname*.bin subdirectory of the application directory.

Once the link step completes successfully, the build process copies any files required for stand-alone execution into the executable directory specified here. During the build, each copied file is reported in the Build status pane.

Note: If an executable directory is not specified for the application, the executable files are not copied.

In addition to generating the DLL and EXE files for the application library, Aion also copies the DLL for each included library, and searches the application for references to external DLLs (in the "Library" property of an external method). As a result, a directory is created that contains all DLLs required for successful execution of the built executables.

Aion will find all references to external DLLs when searching the Library property of external methods. Some of the external DLLs may not be called by the specific logic of the application and are therefore not needed by the application.

You can remove the unneeded DLLs, reducing the number of DLLs delivered with the built application, and therefore reducing the size, and increasing the performance, of the application.

Note: Some of the DLLs and EXEs copied by Aion BRE may be Windows System Libraries. These Libraries should be available on the User's system and therefore do not need to be copied over.

Note: Since some methods can be invoked dynamically, there is no way for the Aion build process to reliably determine which external DLLs may not be used by the application.

Build Directives

You can use the Build Directives page to specify an Interface Layer or filenames for the generated files.

For step-by-step procedures for specifying an interface layer or filenames for the generated files, see Build Directives in the CA Aion BRE online help.

Build the Application

Once you have specified Library Properties and any desired Build Directives, you are ready to build the application.

For step-by-step procedures for building the application, see Building the Application in the CA Aion BRE online help.

Stop the Build

At any time during the build, you can stop building by choosing Stop Build from the Aion File menu. This menu option is only enabled when a build is in progress.

Note: If a build is canceled, the state of the intermediate files cannot be guaranteed; therefore, for the next build you may want to use the auto or clean options (specified in the Settings dialog-choose Settings from the File menu).

Use Interface Layers

As object technology continues to mature, more software applications are being built by combining different object types. For example, an application might be written in Visual Basic, but use a DLL written in C to perform certain logical operations, and may invoke an ActiveX control for a specific user interaction. You can write Aion BRE applications that use objects packaged using a variety of interfaces (for example, C or ActiveX).

In addition, Aion BRE applications or libraries can be generated using different interface layers, which enables the Aion BRE application or library to be used or embedded in some other application. Interface layers are used when generating Aion BRE applications for use in other applications. When you specify an Interface Layer, Aion generates additional code to essentially “wrap” the application's method logic so that it conforms to the appropriate calling conventions.

Select an Interface Layer

If you do not explicitly specify an interface layer, the resulting DLL is only usable by other Aion libraries or applications. Information about the COM Interface Layer is located in the chapter on the COM component. Aion BRE applications can be built and deployed in a number of object formats, including:

- Aion BRE DLL (the default)
- C DLL
- C++ DLL
- Managed C++ DLL (.NET assembly; only deployable in .NET environment)
- Java
- COM DLL (in-process)
- COM EXE (out-of-process)
- Microsoft Transaction Server
- ActiveX control
- MVS COM Client
- CICS COBOL
- CICS C
- CICS PL/I
- IMS COBOL
- IMS C
- IMS PL/I
- MVS COBOL
- MVS PL/I
- TCP/IP Deployment

For step-by-step procedures for specifying an interface layer, see [Selecting an Interface Layer](#) in the CA Aion BRE online help.

Deploy the Application

This section describes the files you will need to install the compiled Aion BRE application.

After building the application, the resulting executable file is relatively small (since most of the code is stored in the .DLL files that will ship with the application). The proportions are analogous to the difference between the physical size of a card catalog and the library of books. The application needs the following files:

- The executable file (.EXE)
- .DLL files (including one for each library included in the application)
- Any .DLL files containing the implementation of external methods

More Information:

[Specify the Executable Directory](#) (see page 442)

Appendix A: CA Aion Business Rules Expert with UML Modeling Software

This appendix explains how to use CA Aion BRE with UML modeling software, which allows you to transfer and share models between two products. To do this, both CA Aion BRE and a UML modeling tool must be installed, but they do not have to run simultaneously.

UML integration enables you to:

- Use a UML modeling program to specify your application classes.
- Bring UML models into the CA Aion BRE development environment in order to generate code.
- Document Aion BRE applications using the diagramming facilities of UML modeling software.

Integration with UML modeling software is based on the XML Metadata Interchange (XMI) standard. The XMI standard describes the construction of an XML document for model metadata interchange between tools. A UML model can be exported to an XMI file, which then can be imported into CA Aion BRE. Similarly, an Aion BRE application can be exported as an XMI file and then imported into UML modeling software.

Note: For information about using UML modeling software, consult the documentation provided with your modeling product.

Currently, CA Aion BRE supports integration with only CA's UML modeling tool, CA Component Modeler (formerly known as Paradigm Plus). This appendix describes integration with CA Component Modeler 4.1 and greater. Some details may differ for Paradigm Plus 4.0.

Object Types

The following equivalent terms are used for objects in CA Aion BRE and UML:

Aion BRE Object	UML Object
Class	Class
Method	Operation

Aion BRE Object	UML Object
Attribute (All datatypes except Pointer)	Attribute
Attributes of type Pointer to... (Note 1)	Association ends (Note 1)

Note 1: See further explanation in the section Associations, Association Ends, and Pointers.

Transferred Properties

When you transport UML models between UML modeling software and CA Aion BRE, only certain object properties are transferred with the model:

Object Type	Exported/Imported Properties
Class/Interfaces	Name Comments (Note 1) Parent Class
Method/Operation	Name Input/Output arguments (Note 2) Return Type (Note 2) Comments (Note 1) Scope (Public/Private/Protected) Class Method
Attribute	Name Comments (Note 1) Type (Note 3) Scope Class Attribute (static attribute in UML) Constant Attribute Initial Value

Note 1: See the information for your particular UML modeling tool. If you use CA Component Modeler, comments are transferred using the Description tab of the Properties window.

Note 2: Method parameters and return values of type Pointer to a class in CA Aion BRE become references to the class in UML.

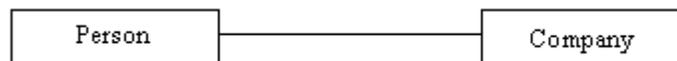
Note 3: Attributes of a type defined by a class in UML become class containment in CA Aion BRE.

Associations, Association Ends, and Pointers

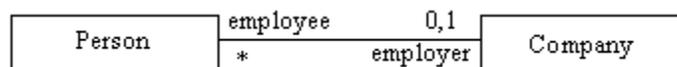
Functionality of a UML model is not expressed merely through class generalizations; in fact, generalizations have little to do with functionality. The principal expression of functionality in a UML model is contained in how classes are related to one another and refer to each other. The principal type of relationship between classes is the **association**. Associations represent reciprocal attributes between two classes.

Associations in UML

UML does not accommodate a data type of Pointer-to that allows attributes of a class to refer to instances of another class. Instead, UML provides associations. An association is a link between two (or more) classes that involves connections among their instances. Consider two classes: Person and Company. To capture the relationship of employment between persons and companies in UML, we represent an association between the classes by drawing a line between them:



We then typically label the ends of these associations (where the association line touches the class) with role names and multiplicities (to show how many instances of the one class can participate in the association with an instance of the other class). The completed relationship may be modeled in UML as:



According to the previous model, a Person performs the role of employee in this association and can have only one Company as employer (or be unemployed). Companies, however, may have many Persons ("*") as their employees.

Correctly modeling this relationship in UML requires careful consideration of which association end gets which role name and multiplicity. In UML, a role is defined as the "face" that a class in an association presents to the class at the opposite end of the association. Thus, the employer role is the face (interface) that a Company presents to Persons in this association. Similarly, employee is the face that Persons present to Companies. Roles go with the class performing that role in the association. Multiplicities follow the role, for example, there is zero or one employer in an Association with each Person instance, and there are zero to many employees for each Company instance.

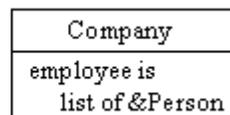
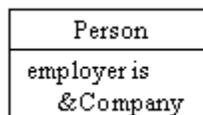
In programming, it is often convenient to use a Pointer to instances of one class without a reciprocal attribute in that other class. This situation is called a *directed association* in UML. It is represented by a directed arrow toward the class being referred to. The association end on the pointing class is designated as non-navigable.

Association Classes

UML also supports the concept of an *association class*. While related to associations, association classes provide a way of conceptualizing the association itself as a class. Association classes are most closely aligned with `_Associations` in CA Aion BRE. It is necessary to observe a strict separation between associations and association classes. Unless otherwise explicitly mentioned, association classes are considered as part of the current discussion.

Map Association Ends to Pointers

It is important to put the role with the correct association end in order to have the association correctly recreated within CA Aion BRE. The role names become the names of attributes in the classes participating in the association. These are attributes of the type Pointer to the class at the other end of the association. However, because a role is only a "face", and it is the class at the other end of the association that "sees" this face, the role name becomes an attribute of the class at the opposite end of the association. Roles cross to the other end when they become attributes in implementation classes. Thus, the preceding association between Person and Company would be represented by classes and Pointer attributes in CA Aion BRE as:



(This crossover occurs when implementing a UML association in any object-oriented language. The only difference between CA Aion BRE and some other object-oriented languages is that not all object-oriented languages use explicit pointer data types for such references.)

If rolenames had not specified, CA Aion BRE would have generated default names of the form:

```
pCompany is &Company  
pEmployee is list of &Person
```

Note: While the “employee” role (singular) is correct given the semantics of UML, it is more desirable to use “employees” for the name of the attribute in CA Aion BRE. It may be considered a matter of preference in UML to use “employees” for the rolename rather than “employee”.

CA Aion BRE maps the multiplicities specified in the UML into either single value attributes of type Pointer-to or to a list of pointers. Multiplicities of zero or one (for example, 1, 0..1, or 0,1) are mapped to single value attributes whose value may include NULL. All other multiplicities whose high end is greater than 1 are mapped to lists. This mapping implies that constrained multiplicities, such as 0..3, will not be explicitly preserved as such in CA Aion BRE. A multiplicity of 0..3 will be a list of pointers of unrestricted membership (cardinality).

Except for the case of Class Containment, all multiplicities, when mapped into CA Aion BRE or exported from CA Aion BRE, will assume the possibility of zero cardinality. That is, all multiplicities will be treated in CA Aion BRE as either 0..1 (single value attribute) or 0..* (for lists).

Note: Class Containment may be specified in the UML by using the name of the contained class as the datatype of an attribute. This is only a temporary solution for representing class containment in UML; although the syntax matches CA Aion BRE's syntax for class containment, the semantics of the UML expression does not match the semantics of class containment in CA Aion BRE. For the appropriate UML representation of class containment, see the section Class Containment in the chapter “Aion Objects Overview”.

Import and Export UML Models

UML models can be imported into CA Aion BRE. CA Aion BRE can also produce UML models that can be imported into a UML modeling tool.

Import UML Models

For step-by-step-procedures for importing UML models, see Importing a UML Model in the CA Aion BRE online help.

For information about which properties are preserved on import, see Transferred Properties.

For information on importing UML models that have been exported from CA Aion BRE into a UML modeling tool, see the documentation for your UML modeling tool (for example, CA Component Modeler).

More Information:

[Transferred Properties](#) (see page 448)

[CA Component Modeler](#) (see page 454)

Merge UML Models with Aion Applications

During the Aion application development process, it is frequently necessary to go back and reconsider the model of the application in the modeling tool. This process typically leads to making incremental changes in the UML model that are to be fed back into, or merged, with the existing Aion application.

To make incremental model changes to the Aion application, you can export the Aion application and re-import it into your UML modeling tool (for example, A CA Component Modeler). Since you are exporting the entire Aion application, which has probably grown since its initial design and importation into Aion, you should replace the original model with the one generated by CA Aion BRE. Make the required changes to the model, then export the new model and import it into your Aion application.

By expanding the class trees in the Import dialog, the programmer can now select those classes to import that have changed.

Note: Once a class is selected, all of its subclasses are automatically selected for importation.

CA Aion BRE matches imported classes and existing classes in the Aion application file based on name and parentage. The following rules apply:

- If the name and parentage of imported class exactly matches the name and parentage of an existing class, CA Aion BRE copies the methods and attributes of the imported class into the existing class.

Note: Currently, CA Aion BRE does not attempt to make any judgments on whether an imported method or attribute is the same or different than an existing one. If a matching method or attribute name already exists within the class, CA Aion BRE will copy the imported method or attribute into the class and append the name with an underscore, “_”. The method signature or attribute type is not considered in this match. It is left to the programmer to decide what to do about the matching methods and attributes that result from a class merger. In most cases, the appropriate action is simply to delete the method or attribute with the underscore.

Note: Merging methods and attributes in this fashion does not provide a distinction between existing methods and attributes that have been deleted from the model in the modeling tool but have not yet been deleted in the Aion application and new methods and attributes that were added to the model in the modeling tool. Distinguishing these in the merged class is left to the programmer.

- If an imported class matches in name to an existing class but the parentage does not match, the importation of that class (and any subclasses) will be failed by the CA Aion BRE importer. If a failed class is truly to be imported, it is necessary for the programmer to make proper adjustments in the existing Aion application, such as deleting or renaming the existing class, and reimporting the class. If an existing class or hierarchy is to be included under a new parent and existing methods should be kept for the sake of retaining their implementations, first cut and paste the existing class or hierarchy to the new parent before importing it.

Export to UML Models

You can export CA Aion BRE libraries as UML models that can be used by CA Component Modeler.

Note: Only certain properties are exported. For information, see Transferred Properties.

For step-by-step-procedures for exporting Aion libraries to UML models, see Exporting to a UML Model in the CA Aion BRE online help.

For information on exporting UML models from a UML modeling tool, see the documentation for your UML modeling tool (for example, CA Component Modeler).

CA Component Modeler

If one of the methodologies supported by CA Component Modeler is used, the level of ImplementationModel should be used to model an Aion application, because CA Aion BRE is an implementation language. CA Aion BRE models should be created as packages contained within an implementation model. A package should be thought of as mapping to a library in the CA Aion BRE implementation.

Important! When exporting a model from CA Component Modeler for use with CA Aion BRE, you should *not* select the XMI 1.1 Compatibility option in the export wizard. Accept the default XMI produced by CA Component Modeler.

Libraries exported from CA Aion BRE can be imported into either models or packages in CA Component Modeler.

It is highly recommended that libraries be imported into packages. Observe the following guidelines as standard procedure for using CA Component Modeler import facilities:

- In general, the import options of Import and Generalize will not be helpful when working the CA Aion BRE libraries.
- When importing a CA Aion BRE library as a model or package, use the import options of Merge or Replace. Using the import option of Containment may not provide all the functionality you wish.

Note: Importing the CA Aion BRE model will rename the UML model or package with the name of the Aion application.

It is typical that an Aion application should use other Aion applications as included libraries. There are several approaches to modeling the library structure of an Aion application with packages in CA Component Modeler. The simplest is package containment. Two packages are required, one contained under the other.

Note: When importing CA Aion BRE libraries into CA Component Modeler, these packages must be created before the import.

The containing package represents the included library of the Aion application. The contained package is the Aion application.

You can create inheritance relationships between classes in the containing package and those in containing package. This type of relationship is preserved when the contained package is exported from CA Component Modeler and will be recognized when it is imported by CA Aion BRE. If the Aion application includes the appropriate library corresponding to the containing package, a similar inheritance relationship will be automatically established in CA Aion BRE.

Note: CA Component Modeler supports a hierarchy of included libraries (contained containing packages); however, this hierarchy can only be a single hierarchy. That is, the Aion application can include at most one library. To represent Aion library inclusions in which an Aion app includes more than one library requires CA Component Modeler's more sophisticated package management facilities, such a publication or the package permission relationship.

Data Type Mapping

CA Component Modeler supports dynamic creation of data types. It is suggested that you use data type names that correspond to the data types in CA Aion BRE when you specify the data type of an attribute or specify the signature of a method.

You may define data types of type list of string or array of integer.

Note: Maintain consistent capitalization when dynamically specifying data types. CA Component Modeler is case sensitive, so that "integer" and "Integer" are treated as two different data types.

The following section describes how to construct elements in your UML model that correspond to specific structures in CA Aion BRE.

Modeling _Interfaces

CA Aion BRE provides `_Interfaces` in order to define pure abstract interfaces for classes. It is possible to model `_Interfaces` directly in a UML model in CA Component Modeler. To create an abstract interface, select the Interface icon from the palette (the Interface icon looks like a lollipop that has fallen on its side) and drag it onto your class model. In CA Component Modeler, Interfaces are treated exactly like `_Interfaces` in CA Aion BRE, in that the UML Interface allows you to specify only method signatures.

To relate the interface to a class that implements the interface, it is necessary to draw a realization abstraction relationship between the class and the interface. Select the Abstraction relationship from the palette and draw the relationship from the class to the interface. Choose the "Realization" stereotype from the pop-up menu that results when you connect the relationship to the target (the interface). You should see the relationship as a dashed line with a generalization arrowhead and a stereotype of `<<realize>>`. The CA Aion BRE importer uses this information to populate the Implements list for the class.

Note: Make sure that you use the realization abstraction relationship between the class and the interface. Any other relationship will be ignored by the CA Aion BRE importer.

As in CA Aion BRE, you should specify the methods defined on the Interface as members of the implementing class as well. Failure to do this will create Invalids in Aion when the class and its interface are imported into CA Aion BRE.

Standards

The following standards should be observed when using CA Aion BRE with a UML modeling tool:

- Always provide unique association end names (role names) for the navigable ends of an association in a UML model. These names are used as the names of the instance pointer attributes in CA Aion BRE and must be unique within an Aion application. This standard applies as well to directed associations; the navigable end must have a name.
- Avoid using UML aggregation (the open diamond) and composition (the black diamond). These structures do not translate into any constructs within CA Aion BRE.

- Use only multiplicities of 0,1 (zero to one) and 0..* (zero to many) on association ends. These are supported in CA Aion BRE.
- In CA Aion BRE, avoid using class-level attributes that define an association (that is, a class attribute of type Pointer-to an instance of some class). Such constructs are not supported in UML; role names cannot be defined as static.
- Do not use spaces in names in UML models that are intended to be imported into CA Aion BRE
- Use consistent capitalization for data types throughout a CA Aion BRE application. Differently capitalized data types in CA Aion BRE yield distinct data types in CA Component Modeler.

Appendix B: Aion--Acknowledgements

This appendix provides acknowledgements for CA and third-party software used with CA Process Manager.

Apache Software License, Version 2.0

Portions of this product include software developed by the Apache Software Foundation (<http://www.apache.org/>). The Apache software is distributed in accordance with the following license agreement.

Apache License

Version 2.0, January 2004

<http://www.apache.org/licenses/>

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

Apache Software License, Version 1.1

This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>). The Apache software is distributed in accordance with the following license agreement.

The Apache Software License, Version 1.1

Apache Ant 1.5.3

Copyright (C) 2000-2003 The Apache Software Foundation. All rights reserved.

Apache Axis 1.1

Copyright (c) 2002 The Apache Software Foundation. All rights reserved.

Apache Cactus 1.5

Copyright (c) 2001-2003 The Apache Software Foundation. All rights reserved.

Apache Jakarta-Oro 2.0

Copyright (c) 2000-2002 The Apache Software Foundation. All rights reserved.

Apache Log4j 1.2.8

Copyright (c) 1999 The Apache Software Foundation. All rights reserved.

Apache Tomcat 4.1.29

Copyright (c) 1999, 2000 The Apache Software Foundation. All rights reserved.

Apache Xalan C++ 1.6

Copyright (c) 1999 The Apache Software Foundation. All rights reserved.

Apache Xalan Java 2.5.2

Copyright (c) 1999-2003 The Apache Software Foundation. All rights reserved.

Apache Xerces C++ 2.3

Copyright (c) 1999-2001 The Apache Software Foundation. All rights reserved.

Apache Xerces Java 2.6

Copyright (C) 1999-2003 The Apache Software Foundation. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

The end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>)."
Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear.

The names "Ant", "Axis", "Cactus", "The Jakarta Project", "Jakarta-Oro", "log4j", "Tomcat", "Xalan", "Xerces", "Apache" and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org.

Products derived from this software may not be called "Apache" or "Jakarta-Oro", nor may "Apache" or "Jakarta-Oro" appear in their name, without prior written permission of the Apache Software Foundation.

THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Apache Ant, Axis, Cactus, Jakarta-Oro, Log4J and Tomcat consist of voluntary contributions made by many individuals on behalf of the Apache Software Foundation. For more information on the Apache Software Foundation, please see <<http://www.apache.org/>>.

Portions of Apache Jakarta-Oro are based upon software originally written by Daniel F. Savarese. We appreciate his contributions.

Apache Xalan C++ and Xalan Java consist of voluntary contributions made by many individuals on behalf of the Apache Software Foundation and were originally based on software copyright (c) 1999, Lotus Development Corporation, <http://www.lotus.com>. For more information on the Apache Software Foundation, please see <<http://www.apache.org/>>.

Apache Xerces C++ and Xerces Java consist of voluntary contributions made by many individuals on behalf of the Apache Software Foundation and were originally based on software copyright (c) 1999, International Business Machines, Inc., <http://www.ibm.com>. For more information on the Apache Software Foundation, please see <<http://www.apache.org/>>.

BusinessObjects Software License

Terms and Conditions for the Use of

Business Objects Software Limited - BusinessObjects Enterprise XI Release 2

BUSINESS OBJECTS LICENSE AGREEMENT

IMPORTANT-READ CAREFULLY: THIS IS A LEGAL AGREEMENT BETWEEN YOU AND BUSINESS OBJECTS FOR THE BUSINESS OBJECTS SOFTWARE PRODUCT ACCOMPANYING THIS AGREEMENT, WHICH MAY INCLUDE COMPUTER SOFTWARE, ASSOCIATED MEDIA, PRINTED MATERIALS AND ONLINE OR ELECTRONIC DOCUMENTATION (?SOFTWARE?). BEFORE CONTINUING WITH THE INSTALLATION OF THE SOFTWARE, YOU MUST READ, ACKNOWLEDGE AND ACCEPT THE TERMS AND CONDITIONS OF THE SOFTWARE LICENSE AGREEMENT THAT FOLLOWS (?AGREEMENT?). IF YOU DO NOT ACCEPT THE TERMS AND CONDITIONS OF THE AGREEMENT, YOU MAY RETURN, WITHIN THIRTY (30) DAYS OF PURCHASE, THE SOFTWARE TO THE PLACE YOU OBTAINED IT FOR A FULL REFUND.

1. GRANT OF LICENSE. Business Objects grants you a nonexclusive and limited license to use the Software products and functionalities for which you have paid the applicable fees solely for your internal business purposes and in accordance with the terms and conditions of this Agreement. The Software is licensed, not sold, to you. If you acquired this product as a special offer or as a promotional license included with another Business Objects product, additional restrictions apply as set forth in section 3.9 below. If you acquired this product bundled or in combination with a third party product, you may only use the Software with the third party product as described in section 3.6 (?Restricted License?) below. This license does not apply to any other software program provided with the Software, including promotional software, which is governed by the online software license agreement included with that software.

?Business Objects? is the Business Objects company from whom you are purchasing the Software or related services, either directly or indirectly through a reseller.

2. INSTALLATION AND USE. You may install and use the Software only in the configuration and for the number of licenses acquired by you. You may also install non-production copies of the Software as is reasonably necessary for disaster recovery, emergency restart and backup, including, but not limited to making copies for such purposes for use at one or more disaster recovery sites. In order to exercise your rights to the Software under this License Agreement you must activate your copy of the Software in the manner described during the launch sequence. Business Objects may control the number and type of licenses and the use of the Software by key codes.

3. LICENSE TYPES AND DEFINITIONS.

3.1. Named User License (?NUL?). When the Software is licensed on a Named User basis, each individual Named User must be specifically identified as the sole holder of a NUL. The sharing of the NUL by more than one individual is expressly prohibited. In addition, NUL(s) may not be transferred from one individual to another unless the original end user no longer requires, and is no longer permitted, access to the Software. NUL(s) are assigned to a single Deployment and may not be shared among different Deployments.

3.2. Concurrent Access License (?CAL?). When the Software is licensed on a Concurrent Access basis, the aggregate number of end users accessing the Software at any one time may not exceed the number of CALs you have obtained. CAL(s) are assigned to a particular Deployment, and may not be shared among different Deployments. When using Concurrent Access licenses, you may not utilize a program or system to cache or queue report requests.

3.3. Processor License. When the Software is licensed on a Processor basis, the aggregate number of central processing units (?Processors?) running any Software components(s) (except the Web Connector, SDK, Report Publishing Wizard and report viewers) may not exceed the number of Processors licensed. A multi-core chip Processor with N processor cores shall be counted as N Processors.

3.4. Server License (Data integration products only). When the Software is licensed on a Server basis, the Software may be loaded onto a single computer with up to four Processors.

3.5. LPAR License (DB2 Information Integrator only). A LPAR license permits use of the Software with a single data source on a single logical partition as implemented by IBM.

3.6. Restricted License. If you acquired the Software bundled or otherwise provided in combination with or for use with a third party product (?OEM Application?), you have acquired a Restricted License. You may use each licensed copy of the Software only in conjunction with the OEM Application with which it was provided. Accessing data that is not specifically created or processed by the OEM Application is in violation of this license. If the OEM Application requires the use of a data mart or data warehouse, the Software may be used with the data mart or data warehouse only to access data created or processed by the OEM Application. Restricted Licenses may not be combined with unrestricted licenses in the same Deployment.

3.7. Development License. If you receive a Development License, you may use the number and type of licenses acquired only to develop or test Deployments. A Development License cannot be used in or transferred to a production environment.

3.8. Update License. If you received the Software as an update to a previously licensed product, your license to use the Software is limited to the aggregate number of licenses you have acquired for the previous product. If you choose to use the Software and the previous product simultaneously, the aggregate number of licenses used to access the Software and the previous product may not exceed the aggregate number of licenses you acquired for the previous product.

3.9. Promotional License. If you received the Software as a special offer or promotional license (?Promotional License?), you may only use the Promotional Licenses with a new Deployment. Promotional Licenses may not be added to or used with an existing Deployment or Project.

3.10. Evaluation/Not for Resale License. An Evaluation or Not For Resale license may be used only for the number and type of licenses specified and for the period specified on the Software packaging, ordering or shipping documentation. If the ordering or shipping documentation specifies a particular project, the Software may be used only with that project. An Evaluation License may only be used for evaluation purposes and may not be used for production purposes.

Notwithstanding any other provision of this Agreement, Software provided under an Evaluation or Not for Resale licenses is provided ?AS-IS? without warranty of any kind, express or implied. An Evaluation License or Not for Resale License may be terminated by Business Objects upon written notice at any time.

3.11. Definitions. ?Deployment? means a single installation of no more than one of the following Software modules: Repository, Security Domain, Central Management Server (?CMS?) or CMS Cluster. ?Project? means one or more Deployments (a) providing the same or substantially similar reports; (b) utilizing the same or a substantially similar custom application interface; or (c) used with applications consisting of related modules or components.

4. PRODUCT SPECIFIC USE RIGHTS.

4.1. Performance Management Application Modules/Solutions. The software components, tools and utilities supplied with a Performance Management Application Module/Solution may only be used with the product with which they were provided.

4.2. BusinessObjects Enterprise. You may not combine licenses for different editions of BusinessObjects Enterprise in a single Deployment (for example, Premium licenses may not be combined with Professional licenses in the same Deployment). You may use BusinessObjects Enterprise Professional to publish and distribute only one of Business Objects' proprietary report format types (Crystal Reports, OLAP Intelligence/Crystal Analysis, Web Intelligence/Desktop Intelligence). Web Intelligence and Desktop Intelligence are deemed a single proprietary report format for this purpose. If you wish to publish and distribute more than one report format type, you must acquire BusinessObjects Enterprise Premium.

4.3. BusinessObjects Enterprise Professional and Crystal Enterprise Professional Options. BusinessObjects Enterprise Professional Options are licensed as add-ons to a Deployment. Options include Crystal Reports Explorer, Auditing, Publishing, Live Office, Integration Kits for third party applications and other products designated as BusinessObjects Enterprise Options. The number and type of Option licenses must match the number and type of BusinessObjects Enterprise licenses in the Deployment in which the Options are used.

4.4. Web Intelligence Interactive Viewing. Keycodes to Web Intelligence Interactive Viewing unlock all features of the full Web Intelligence product. However, Web Intelligence Interactive Viewing is a limited license and may not be utilized to edit or create documents.

4.5. BusinessObjects Rapid Marts. When licensing BusinessObjects Rapid Marts, a license for BusinessObjects Data Integrator must also be obtained. If BusinessObjects Rapid Marts are licensed with BusinessObjects Data Integrator, an individual BusinessObjects Rapid Marts license must be obtained for each BusinessObjects Data Integrator license. Copying one BusinessObjects Rapid Marts license and then deploying it to other instances is prohibited. In addition to the foregoing, you must license certain applicable application interfaces.

4.6. BusinessObjects Data Integrator. If you desire to deploy a Server License to access enterprise data sources such as packaged applications, mainframes, or technology infrastructure products (?Enterprise Data Sources?), you must obtain individual BusinessObjects Data Integrator Interface licenses.

4.7. BusinessObjects Data Integrator Interfaces. When licensing the BusinessObjects Data Integrator Interfaces, licenses for BusinessObjects Data Integrator must also be obtained. An individual interface license must be acquired for each BusinessObjects Data Integrator license. If multiple instances of an Application, Technology, or Mainframe type are accessed by the BusinessObjects Data Integrator Interface, then one interface license must be acquired for each instance. If multiple instances of a Database type are accessed by the BusinessObjects Data Integrator Interface, then only one interface license must be acquired for that Database type. Unlike other Interfaces, Database interfaces are charged per database type and not per instance.

4.8. BusinessObjects Knowledge Accelerator. BusinessObjects Knowledge Accelerator may be used to meet your employee training needs for the number of employees identified to Business Objects (?Employees?) and may not be used by or on behalf of any third party. You shall purchase additional licenses equal to the number of additional or new Employees to be trained. Any customization tools included with the BusinessObjects Knowledge Accelerator Product (RWD Info Pak Simulator, Publisher and Web Architect) shall be used only for modifying or customizing the content developed by BusinessObjects Knowledge Accelerator Product, and only by the number of instructional designers and administrators specified in the sales order. You shall not modify, reverse engineer, or distribute for commercial or non-commercial use of such tools, or use such tools to develop other content, including content related to other Business Objects products. A Named User License of Knowledge Accelerator may not be transferred to another individual unless the original Named User is no longer employed by You.

4.9. BusinessObjects Publisher. BusinessObjects Publisher may be licensed on a: 1) Processor basis, or 2) Named User basis, where each recipient of a report generated by BusinessObjects Publisher must have a Named User license.

5. OWNERSHIP. Business Objects and/or its suppliers retain all right, title and interest in and to the Software and all copies at all times, regardless of the form or media in or on which the original or other copies may subsequently exist. You neither own nor hereby acquire any claim or right of ownership to the Software or to any related patents, copyrights, trademarks or other intellectual property. You agree to retain the Software, the terms of this Agreement as well as any Software benchmark or similar tests (whether performed by you, Business Objects or any third party) in confidence and prevent them from unauthorized disclosure or use except with Business Objects' prior written consent. Business Objects and/or its suppliers reserve all rights not expressly granted to you. Business Objects' suppliers are the intended third party beneficiaries of this License Agreement and have the express right to rely upon and directly enforce the terms set forth herein.

6. COPYRIGHT. The Software is copyrighted by Business Objects and/or its suppliers and is protected by United States copyright and patent laws and international treaty provisions. You may not copy the Software except: (a) to provide a non-production backup copy; or (b) to install the Software components licensed by you, as set forth in Sections 2, on to computers as part of executing the Software. Solely with respect to the documentation included with the Software, you may make a reasonable number of copies (either in hardcopy or electronic form), provided that such copies shall be used only by licensed end users in conjunction with their use of the Software and are not republished or distributed to any third party. You must reproduce and include all copyright notices, trademarks or other proprietary legends of Business Objects and its suppliers on any copy of the Software or documentation made by you. Any and all other copies of the Software made by you are in violation of this Agreement.

7. RESTRICTIONS. Except as expressly permitted by this License Agreement or by applicable law you may not: (a) lease, loan, resell, assign, sublicense, or otherwise distribute the Software or any of the rights granted by this License Agreement without the express written permission of Business Objects; (b) use the Software to provide or operate Application Service Provider (ASP), service bureau, marketing, training, outsourcing services, or consulting services, or any other commercial service related to the Software or to develop training materials; (c) modify (even for purposes of error correction), adapt, or translate the Software or create derivative works therefrom except as necessary to configure the Software using the menus, options and tools provided for such purposes and contained in the Software; (d) in any way reverse engineer, disassemble or decompile the Software or the .RPT report file format (including reverse compiling to ensure interoperability) or any portion thereof except to the extent and for the express purposes authorized by applicable law notwithstanding this limitation; (e) use the Software to develop a product which is competitive with any Business Objects product offerings; (f) use the Software to develop a product that converts the report file (.RPT) format to an alternative report file format used by any general-purpose report writing, data analysis or report delivery product that is not the property of Business Objects; (g) use unauthorized keycode(s) or distribute keycode(s); (h) disclose any Software benchmark results to any third party without Business Objects' prior written approval, (i) permit third party access to, or use of the Software except as expressly permitted herein, and (j) distribute or publish keycode(s). If you wish to exercise any right to reverse engineer to ensure interoperability in accordance with applicable law, you shall first provide written notice to Business Objects and permit Business Objects, at its discretion, to make an offer to provide information and assistance reasonably required to ensure Software interoperability with your other products for a fee to be mutually agreed upon (if any).

8. LIMITED WARRANTY AND REMEDY.

(a) Business Objects warrants to you that: (i) for a period of thirty (30) days from delivery of the Software, the Software will substantially conform to the functional description set forth in the standard documentation accompanying the Software; and (ii) for a period of thirty (30) days from delivery the physical media (e.g., CD-ROM), such physical media will be free from defects in materials and workmanship. Any implied warranties on the Software and media are limited to thirty (30) days from delivery, to the extent such warranties cannot be disclaimed under Section 8(c) below. The above warranties specifically exclude defects resulting from accident, abuse, unauthorized repair, modifications, or enhancements, or misapplication. Business Objects does not warrant that the Software will operate uninterrupted or error free. Delivery of additional copies of, or revisions or upgrades to, the Software, including releases provided under Support Services, shall not restart or otherwise affect the warranty period.

(b) Your exclusive remedy for breach of the above-stated limited warranty shall be, at Business Objects' option, either: (i) correction or replacement of the Software with product(s) which conform to the above-stated limited warranty; or (ii) return of the price paid for the Software and termination of this License Agreement with respect to those copies not in compliance. Such remedy shall be provided to you by Business Objects only if you give Business Objects written notice of any breach of the above-stated limited warranty, within thirty (30) days of delivery of the Software.

(c) EXCEPT FOR EXPRESS WARRANTIES STATED IN THIS SECTION 8, BUSINESS OBJECTS AND ITS SUPPLIERS DISCLAIM ALL OTHER WARRANTIES, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTY (I) OF MERCHANTABILITY, (II) OF FITNESS FOR A PARTICULAR PURPOSE, (III) OF NON-INFRINGEMENT OF THIRD PARTY RIGHTS, OR (IV) AGAINST HIDDEN DEFECTS. Some states/jurisdictions do not allow the exclusion of implied warranties, so the above exclusion may not apply to you, and you may have other legal rights that vary from state to state or by jurisdiction. YOU ACKNOWLEDGE THAT IN ENTERING INTO THIS AGREEMENT, YOU HAVE RELIED UPON YOUR OWN EXPERIENCE, SKILL AND JUDGEMENT TO EVALUATE THE SOFTWARE AND THAT YOU HAVE SATISFIED YOURSELF AS TO THE SUITABILITY OF THE SOFTWARE TO MEET YOUR REQUIREMENTS.

9. LIMITATION OF LIABILITY. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, in no event will BUSINESS OBJECTS or its DISTRIBUTORS, SUPPLIERS or AFFILIATES be liable TO you OR ANY THIRD PARTY FOR ANY INDIRECT, SPECIAL, INCIDENTAL, CONSEQUENTIAL OR PUNITIVE DAMAGES, INCLUDING WITHOUT LIMITATION, ANY LOST PROFITS OR REVENUES, LOSS OR INACCURACY OF ANY DATA, OR COST OF SUBSTITUTE GOODS, REGARDLESS OF THE THEORY OF LIABILITY (INCLUDING NEGLIGENCE) AND EVEN IF BUSINESS OBJECTS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. BUSINESS OBJECTS AND ITS SUPPLIERS' AGGREGATE LIABILITY TO YOU FOR ACTUAL DIRECT DAMAGES FOR ANY CAUSE WHATSOEVER SHALL BE LIMITED TO THE SOFTWARE LICENSE FEES PAID BY YOU FOR THE SOFTWARE OR THE FEES PAID BY YOU FOR THE SERVICE DIRECTLY CAUSING THE DAMAGES. THESE LIMITATIONS WILL APPLY NOTWITHSTANDING ANY FAILURE OF ESSENTIAL PURPOSE OF ANY LIMITED REMEDY. THE FOREGOING ALLOCATION OF RISK IS REFLECTED IN THE FEES CHARGED UNDER THIS LICENSE AGREEMENT. Some states/jurisdictions do not allow the limitation or exclusion of liability IN CERTAIN CIRCUMSTANCES INCLUDED IN THIS SECTION, so the above limitation may not apply to you ONLY IN SUCH CIRCUMSTANCES.

10. SUPPORT SERVICES. If you purchased Support Services, Business Objects will provide to you product support services for the Software in accordance with Business Objects then current Support Services terms and conditions. If you purchase Support Services for the Software, you must purchase Support Services for all authorized copies of said Software in your possession.

11. TERMINATION. This Agreement is effective until terminated. You may terminate this License Agreement at any time by providing Business Objects with written notice, provided that you have complied with the return and/or destruction policy set forth below. However, you shall receive a refund of your license fee only if this Agreement is terminated in compliance with Section 8 hereof. If you ordered an Evaluation License for the Software that is time disabled, this Agreement will automatically terminate after the Evaluation Period, and you agree not to avoid, or attempt to avoid, any applicable time limitation. This Agreement may be terminated by Business Objects if: (i) you fail to pay the license fees and other charges set forth at the time of your order; or (ii) you fail to comply with any of the terms and conditions set forth in this Agreement and do not remedy such failure within thirty (30) days after receiving notice thereof. Termination shall not relieve you from your obligation to pay fees that remain unpaid and shall not limit Business Objects from pursuing other available remedies. Upon termination by Business Objects of this Agreement, Business Objects will have no obligation to refund to you any fees paid by you and you agree to waive in perpetuity and unconditionally any and all claims for refunds. Upon any termination of this Agreement, you agree to: (i) immediately cease all use of the Software, including the use and distribution of any Custom Applications incorporating the Software; and (ii) either return the Software to Business Objects or destroy same, and certify to Business Objects, in writing, that all copies and partial copies thereof have been returned or completely destroyed and are no longer being used. Sections 5, 6, 8(c), 9, 11, 12, 13, 14, 15, 17 and 18 shall survive any termination of this Agreement.

12. AUDIT. During the term of this Agreement and for two (2) year after termination or expiration, Business Objects may audit, upon reasonable notice to you and at Business Objects' expense, your books and records to determine your compliance with this Agreement. In the event any such audit reveals that you have underpaid Business Objects by an amount greater than five percent (5%) of the amounts due Business Objects in the period being audited, or that you have knowingly breached any material obligation hereunder, then, in addition to such other remedies as Business Objects may have, you shall pay or reimburse to Business Objects the cost of the audit.

13. GENERAL. If any provision of this Agreement is ruled invalid, such invalidity shall not affect the validity of the remaining portions of this Agreement. This Agreement constitutes the entire agreement between you and Business Objects, and supersedes any prior agreement, whether written or oral, relating to the subject matter of this Agreement. This Agreement may not be modified except by an instrument in writing duly signed by an authorized representative of each of the parties. If you are acquiring the Software on behalf of an entity, you represent and warrant that you have the legal capacity to bind such entity to this Agreement. All terms of any purchase order or other ordering document submitted by you shall be superseded by this Agreement. In the event you and Business Objects have executed a mutually agreed upon a separately executed software license and related services agreement (?MSLA?) and acquired the Software pursuant to such MSLA, the terms of the MSLA may govern your use of the Software and the terms of this Agreement shall be superseded by the MSLA. The product name for the Software is a trademark or registered trademark of Business Objects. Should you have questions concerning this License Agreement, please contact your local Business Objects sales office or authorized reseller, or write to: Business Objects, Attn: Contracts Department, 3030 Orchard Parkway, San Jose, CA 95134.

14. U.S. GOVERNMENT RESTRICTED RIGHTS. The Software is a "commercial item," as that term is defined at 48 C.F.R. 2.101 (Oct. 1995), consisting of "commercial computer software" and "commercial computer software documentation," as such terms are used in 48 C.F.R. 12.212 (Sept. 1995). Consistent with 48 C.F.R. 12.212 and 48 C.F.R. 227.7202-1 through 227.7202-4 (June 1995) (or an equivalent provision, e.g., in supplements of various U.S. government agencies, as applicable), all U.S. Government users acquire the Software with only those rights set forth herein. Manufacturer is Business Objects, 3030 Orchard Parkway, San Jose, CA 95134.

15. EXPORT CONTROLS. You acknowledge that the Software is of U.S. origin. You agree to comply with all applicable international and national laws that apply to the Software, including the U.S. Export Administration Regulations, as well as end-user, end-use and destination restrictions issued by U.S. and other governments.

16. ORDER TERMS. Purchase orders conforming to Business Objects purchase order requirements may be accepted from qualified companies. All pre-printed terms of any purchase order not approved in writing by Business Objects shall have no effect. Payment terms are net-30 days from date of invoice. FOB Business Objects facility. Business Objects specifically disclaims price guarantees of any kind. You are responsible for payment of all applicable sales, use, consumption, VAT, GST, and other taxes and all applicable export and import fees, custom duties and similar charges, excluding taxes based on Business Objects net income.

17. GOVERNING LAW. Except as otherwise preempted by United States federal law, this Agreement is governed by the laws of the State of California, United States, without reference to conflict of laws provisions or the United Nations 1980 Convention on Contracts for the International Sale of Goods and any amendments thereto.

18. COUNTRY UNIQUE TERMS.

If you purchased the Software in any territory specified below (the "Local Territory"), this section sets forth specific provisions as well as exceptions to the above terms and condition. To the extent any provision applicable to the Local Territory (the "Local Provision") set forth below is in conflict with any other term or condition in this agreement, the Local Provision will supersede such other term or condition with respect to any licenses purchased in the Local Territory.

AUSTRALIA:

a) Limited Warranty and Remedy (Section 8): The following is added:

The warranties specified in this Section are in addition to any rights You may have under the Trade Practices Act 1974 or other legislation and are only limited to the extent permitted by the applicable legislation.

b) Limitation of Liability (Section 9): The following is added:

To the extent permitted by law, where Business Objects is in breach of a condition or warranty implied by the Trade Practices Act 1974 or the equivalent State or Territory legislation which cannot be excluded, Business Objects' liability is limited, at Business Objects' sole election: (i) in case of the Software: (a) (i) to repair or replace the goods, or the supply of equivalent goods, or (ii) payment of the cost of such repair or replacement or of acquiring equivalent goods; and (ii) in case of Support Services: (x) re-supply of the Support Services; or (y) the cost of having the services supplied again. In calculating Business Objects' aggregate liability under this Agreement, the amounts paid or the value of any goods or services replaced, repaired, or supplied by Business Objects pursuant to this paragraph shall be included.

c) Governing Law (Section 17): The following replaces the terms of this section in its entirety:

This Agreement is governed by the laws of the State or Territory in which you acquired the Software, without reference to conflict of laws provisions or the United Nations 1980 Convention on Contracts for the International Sale of Goods and any amendments thereto.

BELGIUM AND FRANCE:

a) Limitation of Liability (Section 9): The following replaces the terms of this section in its entirety:

Except as otherwise provided by mandatory law:

1. Business Objects' liability for any damages and losses that may arise as a result of the performance of its obligations in connection with this Agreement is limited to the compensation of only those damages and losses proved and actually arising as an immediate and direct consequence of the non-fulfillment of such obligations (if Business Objects is at fault), for a maximum amount equal to the charges You paid for the Software that has caused the damages. This limitation shall not apply to damages for bodily injuries (including death) and damages to real property and tangible personal property for which Business Objects is legally liable.

2. UNDER NO CIRCUMSTANCES IS BUSINESS OBJECTS, OR ANY OF ITS SOFTWARE DEVELOPERS, LIABLE FOR ANY OF THE FOLLOWING, EVEN IF INFORMED OF THEIR POSSIBILITY: 1) LOSS OF, OR DAMAGE TO, DATA; 2) INCIDENTAL OR INDIRECT DAMAGES, OR FOR ANY ECONOMIC CONSEQUENTIAL DAMAGES; 3) LOST PROFITS, EVEN IF THEY ARISE AS AN IMMEDIATE CONSEQUENCE OF THE EVENT THAT GENERATED THE DAMAGES; OR 4) LOSS OF BUSINESS, REVENUE, GOODWILL, OR ANTICIPATED SAVINGS.

3. The limitation and exclusion of liability herein agreed applies not only to the activities performed by Business Objects but also to the activities performed by its suppliers and Software developers, and represents the maximum amount for which Business Objects as well as its suppliers and Software developers, are collectively responsible. This limitation shall not apply to damages for bodily injuries (including death) and damages to real property and tangible personal property for which Business Objects is legally liable.

b) Governing Law (Section 17): The following replaces the terms of this section in its entirety:

This Agreement is governed by the laws of country in which you acquired the Software, without reference to conflict of laws provisions or the United Nations 1980 Convention on Contracts for the International Sale of Goods and any amendments thereto.

GERMANY AND AUSTRIA:

a) Warranty (Section 8): The following replaces the terms of this section in its entirety:

Business Objects warrants that the Software provides the functionalities set forth in the associated documentation (?Documented Functionalities?) for the Limited Warranty Period following receipt of the Software when used on the recommended hardware configuration. Limited Warranty Period means one year if you are a business user and two years if you are not a business user. Non-substantial variation from the Documented Functionalities does not establish any warranty rights. THIS LIMITED WARRANTY DOES NOT APPLY TO SOFTWARE PROVIDED TO YOU FREE OF CHARGE (FOR EXAMPLE, UPDATES, PRE-RELEASE, EVALUATION, OR NFR) OR SOFTWARE THAT HAS BEEN ALTERED BY YOU, TO THE EXTENT SUCH ALTERATION CAUSED A DEFECT. To make a warranty claim, you must return, at Business Objects expense, the Software and proof of purchase to the company from whom you obtained it. If the functionalities of the Software vary substantially from the agreed upon functionalities, Business Objects is entitled, by way of re-performance and at its own discretion, to repair or replace the Software. If that fails, you are entitled to a reduction of the purchase price or to cancel the purchase agreement.

b) Limitation of Liability (Section 9): the following paragraph is added to this Section:

The limitations and exclusions specified in this Section will not apply to damages caused by Business Objects' intentional or by gross negligence. In addition, Business Objects shall be responsible up to the amount of the typically foreseeable damages from any damage which has been caused by Business Objects or its agents due to the slightly negligent breach of a material contractual duty. This limitation of liability shall apply to all damage claims, irrespective of the legal basis there of and in particular, to any pre-contractual or auxiliary contractual claims. This limitation of liability shall not, however, apply to any mandatory statutory liability under the product liability act nor to any damage which is caused due to the breach of an express warranty to the extent the express warranty was intended to protect you from the specific damage incurred. This clause shall not be intended to limit liability where the extent of liability is provided by mandatory law.

c) Governing Law (Section 17): The following replaces the terms of this section in its entirety:

This Agreement is governed by the laws of country in which you acquired the Software, without reference to conflict of laws provisions or the United Nations 1980 Convention on Contracts for the International Sale of Goods and any amendments thereto.

ITALY:

a) Limitation of Liability (Section 9): the following replaces the terms of this section in its entirety:

Apart from damages arising out of gross negligence or willful misconduct for which Business Objects may not limit its liability, Business Objects' liability for direct and indirect damages related to the original or further defects of the Software, or related to the use or the nonuse of the Software or related to any case whatsoever for breach of the Agreement, shall be limited to the fees paid by you to Business Objects for the Software or for the part of the Software upon which the damages were based.

b) Governing Law (Section 17): The following replaces the terms of this section in its entirety:

This Agreement is governed by the laws of country in which you acquired the Software, without reference to conflict of laws provisions or the United Nations 1980 Convention on Contracts for the International Sale of Goods and any amendments thereto.

UNITED KINGDOM:

c) Governing Law (Section 17): The following replaces the terms of this section in its entirety:

This Agreement is governed by the laws of England and Wales, without reference to conflict of laws provisions or the United Nations 1980 Convention on Contracts for the International Sale of Goods and any amendments thereto.

Notwithstanding any other provision in this Agreement, nothing in this Agreement shall create or confer (whether expressly or by implication) any rights or other benefits whether pursuant to the Contracts Rights of Third Parties) Act 1999 or otherwise in favour of any person not a party hereto.

Please indicate below whether you accept, or do not accept, the terms and conditions of this software license agreement.

CA Trusted Open Source License

CA Trusted Open Source License, Version 1.0

PLEASE READ THIS DOCUMENT CAREFULLY AND IN ITS ENTIRETY. THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS COMPUTER ASSOCIATES TRUSTED OPEN SOURCE LICENSE ("LICENSE"). ANY USE, REPRODUCTION, MODIFICATION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES THE RECIPIENT'S ACCEPTANCE OF THIS LICENSE.

License Background

CA International, Inc. ("CA") believes in open source. We believe that the open source development approach can take appropriate software programs to unprecedented levels of quality, growth, and innovation. To demonstrate our continuing commitment to open source, we are releasing the Program (as defined below) under this License.

This License is intended to permit contributors and recipients of the Program to use the Program, including its source code, freely and without many of the concerns of some other open source licenses. Although we expect the underlying Program, and Contributions (as defined below) made to such Program, to remain open, this License is designed to permit you to maintain your own software programs free of this License unless you choose to do so. Thus, only your Contributions to the Program must be distributed under the terms of this License.

The provisions that follow set forth the terms and conditions under which you may use the Program.

1. DEFINITIONS

1.1 "Contribution" means (a) in the case of CA, the Original Program; and (b) in the case of each Contributor (including CA), changes and additions to the Program, where such changes and/or additions to the Program originate from and are distributed by that particular Contributor to unaffiliated third parties. A Contribution "originates" from a Contributor if it was added to the Program by such Contributor itself or anyone acting on such Contributor's behalf. Contributions do not include additions to the Program which: (x) are separate modules of software distributed in conjunction with the Program under their own license agreement, and (y) are not derivative works of the Program.

1.2 "Contributor" means CA and any other person or entity that distributes the Program.

1.3 "Contributor Version" means as to a Contributor, that version of the Program that includes the Contributor's Contribution but not any Contributions made to the Program thereafter.

1.4 "Larger Work" means a work that combines the Program or portions thereof with code not governed by the terms of this License.

1.5 "Licensed Patents" mean patents licensable by a Contributor that are infringed by the use or sale of its Contribution alone or when combined with the Program.

1.6 "Original Program" means the original version of the software to which this License is attached and as released by CA, including source code, object code and documentation, if any.

1.7 "Program" means the Original Program and Contributions.

1.8 "Recipient" means anyone who modifies, copies, uses or distributes the Program.

2. GRANT OF RIGHTS

2.1 Subject to the terms of this License, each Contributor hereby grants Recipient an irrevocable, non-exclusive, worldwide, royalty-free license to reproduce, prepare derivative works of, publicly display, publicly perform, distribute and sublicense the Contribution of such Contributor, if any, and such derivative works, in source code and object code form. For the avoidance of doubt, the license provided in this Section 2.1 shall not include a license to any Licensed Patents of a Contributor.

2.2 Subject to the terms of this License, each Contributor hereby grants Recipient an irrevocable, non-exclusive, worldwide, royalty-free license to the Licensed Patents to the extent necessary to make, use, sell, offer to sell and import the Contribution of such Contributor, if any, in source code and object code form. The license granted in this Section 2.2 shall apply to the combination of the Contribution and the Program if, at the time the Contribution is added by the Contributor, such addition of the Contribution causes the Licensed Patents to be infringed by such combination. Notwithstanding the foregoing, no license is granted under this Section 2.2: (a) for any code or works that do not include the Contributor Version, as it exists and is used in accordance with the terms hereof; (b) for infringements caused by: (i) third party modifications of the Contributor Version; or (ii) the combination of Contributions made by each such Contributor with other software (except as part of the Contributor Version) or other devices; or (c) with respect to Licensed Patents infringed by the Program in the absence of Contributions made by that Contributor.

2.3 Recipient understands that although each Contributor grants the licenses to its Contributions set forth herein, except as provided in Section 2.4, no assurances are provided by any Contributor that the Program does not infringe the patent or other intellectual property rights of any other person or entity. Each Contributor disclaims any liability to Recipient for claims brought by any other person or entity based on infringement of intellectual property rights or otherwise. As a condition to exercising the rights and licenses granted hereunder, each Recipient hereby assumes sole responsibility to secure any other intellectual property rights needed, if any.

2.4 Each Contributor represents and warrants that it has all right, title and interest in the copyrights in its Contributions, and has the right to grant the copyright licenses set forth in this License.

3. DISTRIBUTION REQUIREMENTS

3.1 A Contributor may choose to distribute the Program in object code form under its own license agreement, provided that:

- a. it complies with the terms and conditions of this License; and
- b. its license agreement:
 - i. effectively disclaims on behalf of all Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose, to the maximum extent permitted by applicable law;
 - ii. effectively excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits, to the maximum extent permitted by applicable law;
 - iii. states that any provisions which are inconsistent with this License are offered by that Contributor alone and not by any other party; and
 - iv. states that source code for the Program is available from such Contributor at the cost of distribution, and informs licensees how to obtain it in a reasonable manner.

3.2 When the Program is made available in source code form:

- a. it must be made available under this License; and
- b. a copy of this License must be included with each copy of the Program.

3.3 If the Program is distributed in object code form, then a prominent notice must be included in the code itself as well as in any related documentation, stating that the source code for the Program is available from the Contributor with information on how and where to obtain the source code.

3.4 This License is intended to facilitate the commercial distribution of the Program by any Contributor. However, Contributors may only charge Recipients a one-time, upfront fee for the distribution of the Program. Contributors may not charge Recipients any recurring charge, license fee, or any ongoing royalty for the Recipient's exercise of its rights under this License to the Program. Contributors shall make the source code for the Contributor Version they distribute available at a cost, if any, equal to the cost to the Contributor to physically copy and distribute the work.

3.5 A Contributor may create a Larger Work by combining the Program with other software code not governed by the terms of this License, and distribute the Larger Work as a single product. In such a case, the Contributor must make sure that the requirements of this License are fulfilled for the Program. Any Contributor who includes the Program in a commercial product offering, including as part of a Larger Work, may subject itself, but not any other Contributor, to additional contractual commitments, including, but not limited to, performance warranties and non-infringement representations on such Contributor's behalf. No Contributor may create any additional liability for other Contributors. Therefore, if a Contributor includes the Program in a commercial product offering, such Contributor ("Commercial Contributor") hereby agrees to defend and indemnify every other Contributor ("Indemnified Contributor") who made Contributions to the Program distributed by the Commercial Contributor against any losses, damages and costs (collectively "Losses") arising from claims, lawsuits and other legal actions brought by a third party against the Indemnified Contributor to the extent caused by the acts or omissions, including any additional contractual commitments, of such Commercial Contributor in connection with its distribution of the Program. The obligations in this section do not apply to any claims or Losses relating to any actual or alleged intellectual property infringement.

3.6 If Contributor has knowledge that a license under a third party's intellectual property rights is required to exercise the rights granted by such Contributor under Sections 2.1 or 2.2, Contributor must (a) include a text file with the Program source code distribution titled "../IP_ISSUES", and (b) notify CA in writing at Computer Associates International, Inc., One Computer Associates Plaza, Islandia, New York 11749, Attn: Open Source Group or by email at opensource@ca.com, both describing the claim and the party making the claim in sufficient detail that a Recipient and CA will know whom to contact with regard to such matter. If Contributor obtains such knowledge after the Contribution is made available, Contributor shall also promptly modify the IP_ISSUES file in all copies Contributor makes available thereafter and shall take other steps (such as notifying appropriate mailing lists or newsgroups) reasonably calculated to inform those who received the Program that such new knowledge has been obtained.

3.7 Recipient shall not remove, obscure, or modify any CA or other Contributor copyright or patent proprietary notices appearing in the Program, whether in the source code, object code or in any documentation. In addition to the obligations set forth in Section 4, each Contributor must identify itself as the originator of its Contribution, if any, in a manner that reasonably allows subsequent Recipients to identify the originator of the Contribution.

4. CONTRIBUTION RESTRICTIONS

4.1 Each Contributor must cause the Program to which the Contributor provides a Contribution to contain a file documenting the changes the Contributor made to create its version of the Program and the date of any change. Each Contributor must also include a prominent statement that the Contribution is derived, directly or indirectly, from the Program distributed by a prior Contributor, including the name of the prior Contributor from which such Contribution was derived, in (a) the Program source code, and (b) in any notice in an executable version or related documentation in which the Contributor describes the origin or ownership of the Program.

5. NO WARRANTY

5.1 EXCEPT AS EXPRESSLY SET FORTH IN THIS LICENSE, THE PROGRAM IS PROVIDED "AS IS" AND IN ITS PRESENT STATE AND CONDITION. NO WARRANTY, REPRESENTATION, CONDITION, UNDERTAKING OR TERM, EXPRESS OR IMPLIED, STATUTORY OR OTHERWISE, AS TO THE CONDITION, QUALITY, DURABILITY, PERFORMANCE, NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE OR USE OF THE PROGRAM IS GIVEN OR ASSUMED BY ANY CONTRIBUTOR AND ALL SUCH WARRANTIES, REPRESENTATIONS, CONDITIONS, UNDERTAKINGS AND TERMS ARE HEREBY EXCLUDED TO THE FULLEST EXTENT PERMITTED BY LAW.

5.2 Each Recipient is solely responsible for determining the appropriateness of using and distributing the Program and assumes all risks associated with its exercise of rights under this License, including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations.

5.3 Each Recipient acknowledges that the Program is not intended for use in the operation of nuclear facilities, aircraft navigation, communication systems, or air traffic control machines in which case the failure of the Program could lead to death, personal injury, or severe physical or environmental damage.

6. DISCLAIMER OF LIABILITY

6.1 EXCEPT AS EXPRESSLY SET FORTH IN THIS LICENSE, AND TO THE EXTENT PERMITTED BY LAW, NO CONTRIBUTOR SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. TRADEMARKS AND BRANDING

7.1 This License does not grant any Recipient or any third party any rights to use the trademarks or trade names now or subsequently posted at <http://www.ca.com/catrdmrk.htm>, or any other trademarks, service marks, logos or trade names belonging to CA (collectively "CA Marks") or to any trademark, service mark, logo or trade name belonging to any Contributor. Recipient agrees not to use any CA Marks in or as part of the name of products derived from the Original Program or to endorse or promote products derived from the Original Program.

7.2 Subject to Section 7.1, Recipients may distribute the Program under trademarks, logos, and product names belonging to the Recipient provided that all copyright and other attribution notices remain in the Program.

8. PATENT LITIGATION

8.1 If Recipient institutes patent litigation against any person or entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Program itself (excluding combinations of the Program with other software or hardware) infringes such Recipient's patent(s), then such Recipient's rights granted under Section 2.2 shall terminate as of the date such litigation is filed.

9. OWNERSHIP

9.1 Subject to the licenses granted under this License in Sections 2.1 and 2.2 above, each Contributor retains all rights, title and interest in and to any Contributions made by such Contributor. CA retains all rights, title and interest in and to the Original Program and any Contributions made by or on behalf of CA ("CA Contributions"), and such CA Contributions will not be automatically subject to this License. CA may, at its sole discretion, choose to license such CA Contributions under this License, or on different terms from those contained in this License or may choose not to license them at all.

10. TERMINATION

10.1 All of Recipient's rights under this License shall terminate if it fails to comply with any of the material terms or conditions of this License and does not cure such failure in a reasonable period of time after becoming aware of such noncompliance. If Recipient's rights under this License terminate, Recipient agrees to cease use and distribution of the Program as soon as reasonably practicable. However, Recipient's obligations under this License and any licenses granted by Recipient as a Contributor relating to the Program shall continue and survive termination.

11. GENERAL

11.1 If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties hereto, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

11.2 CA may publish new versions (including revisions) of this License from time to time. Each new version of the License will be given a distinguishing version number. The Program (including Contributions) may always be distributed subject to the version of the License under which it was received. In addition, after a new version of the License is published, Contributor may elect to distribute the Program (including its Contributions) under the new version. No one other than CA has the right to modify this License.

11.3 If it is impossible for Recipient to comply with any of the terms of this License with respect to some or all of the Program due to statute, judicial order, or regulation, then Recipient must: (a) comply with the terms of this License to the maximum extent possible; and (b) describe the limitations and the code they affect. Such description must be included in the IP_ISSUES file described in Section 3.6 and must be included with all distributions of the Program source code. Except to the extent prohibited by statute or regulation, such description must be sufficiently detailed for a Recipient of ordinary skill to be able to understand it.

11.4 This License is governed by the laws of the State of New York. No Recipient will bring a legal action under this License more than one year after the cause of action arose. Each Recipient waives its rights to a jury trial in any resulting litigation. Any litigation or other dispute resolution between a Recipient and CA relating to this License shall take place in the State of New York, and Recipient and CA hereby consent to the personal jurisdiction of, and venue in, the state and federal courts within that district with respect to this License. The application of the United Nations Convention on Contracts for the International Sale of Goods is expressly excluded.

11.5 Where Recipient is located in the province of Quebec, Canada, the following clause applies: The parties hereby confirm that they have requested that this License and all related documents be drafted in English. Les parties contractantes confirment qu'elles ont exigé que le présent contrat et tous les documents associés soient rédigés en anglais.

11.6 The Program is subject to all export and import laws, restrictions and regulations of the country in which Recipient receives the Program. Recipient is solely responsible for complying with and ensuring that Recipient does not export, re-export, or import the Program in violation of such laws, restrictions or regulations, or without any necessary licenses and authorizations.

11.7 This License constitutes the entire agreement between the parties with respect to the subject matter hereof.

HP Java 2 Runtime Environment 1.4.2

Terms and Conditions for the Use of HP-UX Runtime Environment for the Java™ 2 Platform, Version 1.4

Licensee agrees that the following terms (in addition to the applicable provisions above) shall apply with respect to any open source provided by HP contained within the Product. Notwithstanding anything contained in the CA End User License Agreement, solely with respect to such open source, these terms are not superseded by any written agreement between CA and Licensee:

HP-UX Runtime Environment for the Java™ 2 Platform (the "Software") is owned and copyrighted by HP or its third party suppliers. This license confers no title or ownership in the Software and is not a sale of any rights in the Software. HP's third party suppliers may protect their rights in the event of any violation of these license terms.

Licensee acknowledges that HP may terminate this license for the Software upon notice for failure to comply with any of these license terms. Upon termination, Licensee must immediately destroy the Software, together with all copies, adaptations and merged portions in any form.

By accepting this license agreement, Licensee confirms that it is not located in (or a national resident of) any country under U.S. economic embargo, not identified on any U.S. Department of Commerce Denied Persons List, Entity List or Treasury Department Designated Nationals exclusion list, and not directly or indirectly involved in the development or production of nuclear, chemical, biological weapons or in missile technology programs as specified in the U.S. Export Administration Regulations.

Licensee acknowledges that the Software is not designed or intended for use in on-line control of aircraft, air traffic, aircraft navigation, or aircraft communications; or in the design, construction, operation or maintenance of any nuclear facility. HP disclaims any express or implied warranty of fitness for such uses.

HP does not warrant that the operation of the Software will be uninterrupted or error free. If HP is unable, within a reasonable time, to repair or replace the Software to a condition warranted, Licensee will be entitled to a refund of the purchase price paid by Licensee to HP, which Licensee acknowledges is \$0, upon prompt return of the Software. HP's warranty does not apply to defects resulting from: a) improper or inadequate maintenance of calibration; b) software, interfacing, parts or supplies not supplied by HP; c) unauthorized modification or misuse; d) operation outside of the published environmental specifications for the Software; e) improper site preparation or maintenance, or f) the presence of code from HP suppliers embedded in or bundled with the Software.

TO THE EXTENT ALLOWED BY LOCAL LAW, THE ABOVE WARRANTIES ARE EXCLUSIVE AND NO OTHER WARRANTY OR CONDITION, WHETHER WRITTEN OR ORAL, IS EXPRESSED OR IMPLIED AND HP SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY, SATISFACTORY QUALITY, AND FITNESS FOR A PARTICULAR PURPOSE. Some countries, states, or provinces do not allow limitations on the duration of an implied warranty, so the above limitation or exclusion may not apply to Licensee. This warranty gives Licensee specific legal rights and Licensee might also have other rights that vary from country to country, state to state, or province to province. The foregoing shall not affect any warranties provided in any other applicable agreement between Licensee and CA.

TO THE EXTENT ALLOWED BY LOCAL LAW, THE REMEDIES IN THIS WARRANTY STATEMENT ARE LICENSEE'S SOLE AND EXCLUSIVE REMEDIES. EXCEPT AS INDICATED ABOVE, IN NO EVENT WILL HP OR ITS SUPPLIERS BE LIABLE FOR LOSS OF DATA OR FOR DIRECT, SPECIAL, INCIDENTAL, CONSEQUENTIAL (INCLUDING LOST PROFIT OR DATA), OR OTHER DAMAGE, WHETHER BASED IN CONTRACT, TORT, OR OTHERWISE. Some countries, states, or provinces do not allow the exclusion or limitation of incidental or consequential damages, so the above limitation may not apply to Licensee.

HP-UX Runtime Environment for the Java 2 Platform, Version 1.4

Terms and Conditions for the Use of HP-UX Runtime Environment for the Java(TM) 2 Platform, Version 1.4

Licensee agrees that the following terms (in addition to the applicable provisions above) shall apply with respect to any open source provided by HP contained within the Product. Notwithstanding anything contained in the CA End User License Agreement ("EULA"), solely with respect to such open source, these terms are not superseded by any written agreement between CA and Licensee:

HP-UX Runtime Environment for the Java(TM) 2 Platform (the "Software") is owned and copyrighted by HP or its third party suppliers. This license confers no title or ownership in the Software and is not a sale of any rights in the Software. HP's third party suppliers may protect their rights in the event of any violation of these license terms.

HP may terminate this license for the Software upon notice for failure to comply with any of these license terms. Upon termination, Licensee must immediately destroy the Software, together with all copies, adaptations and merged portions in any form.

By accepting this license agreement, Licensee confirms that it is not located in (or a national resident of) any country under U.S. economic embargo, not identified on any U.S. Department of Commerce Denied Persons List, Entity List or Treasury Department Designated Nationals exclusion list, and not directly or indirectly involved in the development or production of nuclear, chemical, biological weapons or in missile technology programs as specified in the U.S. Export Administration Regulations.

Licensee acknowledges that the Software is not designed or intended for use in on-line control of aircraft, air traffic, aircraft navigation, or aircraft communications; or in the design, construction, operation or maintenance of any nuclear facility. HP disclaims any express or implied warranty of fitness for such uses.

HP does not warrant that the operation of the Software will be uninterrupted or error free. If HP is unable, within a reasonable time, to repair or replace the Software to a condition warranted, Licensee will be entitled to a refund of the purchase price paid by Licensee to HP, which Licensee acknowledges is \$0, upon prompt return of the Software. HP's warranty does not apply to defects resulting from: a) improper or inadequate maintenance of calibration; b) software, interfacing, parts or supplies not supplied by HP; c) unauthorized modification or misuse; d) operating outside of the published environmental specifications for the Software; or e) improper site preparation or maintenance.

TO THE EXTENT ALLOWED BY LOCAL LAW, THE ABOVE WARRANTIES ARE EXCLUSIVE AND NO OTHER WARRANTY OR CONDITION, WHETHER WRITTEN OR ORAL, IS EXPRESSED OR IMPLIED AND HP SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY, SATISFACTORY QUALITY, AND FITNESS FOR A PARTICULAR PURPOSE. Some countries, states, or provinces do not allow limitations on the duration of an implied warranty, so the above limitation or exclusion may not apply to Licensee. This warranty gives Licensee specific legal rights and Licensee might also have other rights that vary from country to country, state to state, or province to province. The foregoing shall not affect any warranties provided in the EULA or any other applicable agreement between Licensee and CA.

TO THE EXTENT ALLOWED BY LOCAL LAW, THE REMEDIES IN THIS WARRANTY STATEMENT ARE LICENSEE'S SOLE AND EXCLUSIVE REMEDIES. EXCEPT AS INDICATED ABOVE, IN NO EVENT WILL HP OR ITS SUPPLIERS BE LIABLE FOR LOSS OF DATA OR FOR DIRECT, SPECIAL, INCIDENTAL, CONSEQUENTIAL (INCLUDING LOST PROFIT OR DATA), OR OTHER DAMAGE, WHETHER BASED IN CONTRACT, TORT, OR OTHERWISE. Some countries, states, or provinces do not allow the exclusion of limitation of incidental or consequential damages, so the above limitation may not apply to Licensee.

IBM 32-bit Runtime Environment for AIX, Java 2 Technology Edition, Version 1.4

Licensee agrees that the following terms (in addition to the applicable provisions above) shall apply with respect to any open source provided by International Business Machines Corporation contained within the Product. Notwithstanding anything contained in the CA End User License Agreement, solely with respect to such open source, these terms are not superseded by any written agreement between CA and Licensee:

The IBM® 32-bit Runtime Environment for AIX™, Java™ 2 Technology Edition, Version 1.4 (the "Program") is owned by International Business Machines Corporation or one of its subsidiaries (IBM) or an IBM supplier, and is copyrighted and licensed, not sold.

No Warranty

SUBJECT TO ANY STATUTORY WARRANTIES WHICH CAN NOT BE EXCLUDED, IBM MAKES NO WARRANTIES OR CONDITIONS EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, THE WARRANTY OF NON-INFRINGEMENT AND THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, REGARDING THE PROGRAM OR TECHNICAL SUPPORT, IF ANY. IBM MAKES NO WARRANTY REGARDING THE CAPABILITY OF THE PROGRAM TO CORRECTLY PROCESS, PROVIDE AND/OR RECEIVE DATE DATA WITHIN AND BETWEEN THE 20TH AND 21ST CENTURIES. This does not affect any warranties contained in any other applicable agreement between Licensee and CA.

The exclusion also applies to any of IBM's subcontractors, suppliers, or program developers (collectively called "Suppliers").

Limitation of Liability

NEITHER IBM NOR ITS SUPPLIERS WILL BE LIABLE FOR ANY DIRECT OR INDIRECT DAMAGES, INCLUDING WITHOUT LIMITATION, LOST PROFITS, LOST SAVINGS, OR ANY INCIDENTAL, SPECIAL, OR OTHER ECONOMIC CONSEQUENTIAL DAMAGES, EVEN IF IBM IS INFORMED OF THEIR POSSIBILITY. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE ABOVE EXCLUSION OR LIMITATION MAY NOT APPLY TO LICENSEE.

General

Licensee acknowledges that IBM may terminate Licensee's license if Licensee fails to comply with the terms of this agreement. If IBM does so, Licensee must immediately destroy the Program and all copies Licensee made of it.

With respect to any claim by or against IBM relating to the Program, neither Licensee nor IBM will bring a legal action more than two years after the cause of action arose unless otherwise provided by local law without the possibility of contractual waiver or limitation.

The laws of the country in which Licensee acquires the Program govern this agreement, except 1) in Australia, the laws of the State or Territory in which the transaction is performed govern this agreement; 2) in Albania, Armenia, Belarus, Bosnia/Herzegovina, Bulgaria, Croatia, Czech Republic, Georgia, Hungary, Kazakhstan, Kirghizia, Former Yugoslav Republic of Macedonia (FYROM), Moldova, Poland, Romania, Russia, Slovak Republic, Slovenia, Ukraine, and Federal Republic of Yugoslavia, the laws of Austria govern this agreement; 3) in the United Kingdom, all disputes relating to this agreement will be governed by English Law and will be submitted to the exclusive jurisdiction of the English courts; 4) in Canada, the laws in the Province of Ontario govern this agreement; and 5) in the United States and Puerto Rico, and People's Republic of China, the laws of the State of New York govern this agreement.

Part 2 - Country-unique Terms

AUSTRALIA:

No Warranty:

The following paragraph is added to this Section:

Although IBM specifies that there are no warranties, Licensee may have certain rights under the Trade Practices Act 1974 or other legislation and are only limited to the extent permitted by the applicable legislation.

Limitation of Liability:

The following paragraph is added to this Section:

Where IBM is in breach of a condition or warranty implied by the Trade Practices Act 1974, IBM's liability is limited to the repair or replacement of the goods, or the supply of equivalent goods. Where that condition or warranty relates to right to sell, quiet possession or clear title, or the goods are of a kind ordinarily acquired for personal, domestic or household use or consumption, then none of the limitations in this paragraph apply.

GERMANY:

No Warranty:

The following paragraphs are added to this Section:

The minimum warranty period for Programs is six months.

In case a Program is delivered without Specifications, IBM will only warrant that the Program information correctly describes the Program and that the Program can be used according to the Program information. Licensee has to check the usability according to the Program information within the "money-back guaranty" period.

Limitation of Liability:

The following paragraph is added to this Section:

The limitations and exclusions specified in the agreement will not apply to damages caused by IBM with fraud or gross negligence, and for express warranty.

INDIA:

General:

The following replaces the second paragraph of this Section:

If no suit or other legal action is brought, within two years after the cause of action arose, in respect of any claim that either party may have against the other, the rights of the concerned party in respect of such claim will be forfeited and the other party will stand released from its obligations in respect of such claim.

IRELAND:

No Warranty:

The following paragraph is added to this Section:

Except as expressly provided in these terms and conditions, all statutory conditions, including all warranties implied, but without prejudice to the generality of the foregoing, all warranties implied by the Sale of Goods Act 1893 or the Sale of Goods and Supply of Services Act 1980 are hereby excluded.

ITALY:

Limitation of Liability:

This Section is replaced by the following:

Unless otherwise provided by mandatory law, IBM is not liable for any damages which might arise.

NEW ZEALAND:

No Warranty:

The following paragraph is added to this Section:

Although IBM specifies that there are no warranties, Licensee may have certain rights under the Consumer Guarantees Act 1993 or other legislation, which cannot be excluded or limited. The Consumer Guarantees Act 1993 will not apply in respect of any goods or services which IBM provides, if Licensee requires the goods and services for the purposes of a business as defined in that Act.

Limitation of Liability:

The following paragraph is added to this Section:

Where Programs are not acquired for the purposes of a business as defined in the Consumer Guarantees Act 1993, the limitations in this Section are subject to the limitations in that Act.

PEOPLE'S REPUBLIC OF CHINA:

Charges:

The following paragraph is added as a new Section:

All banking charges incurred in the People's Republic of China will be borne by Licensee and those incurred outside the People's Republic of China will be borne by IBM.

UNITED KINGDOM:

Limitation of Liability:

The following paragraph is added to this Section at the end of the first paragraph:

The limitation of liability will not apply to any breach of IBM's obligations implied by Section 12 of the Sales of Goods Act 1979 or Section 2 of the Supply of Goods and Services Act 1982.

IBM Java 2 Runtime Environment 1.3.1

Acknowledgment:

CONTAINS IBM Runtime Environment for AIX®, Java™ 2 Technology Edition Runtime Modules

© Copyright IBM Corporation 1999, 2000

All Rights Reserved

Terms and Conditions for the Use of IBM Runtime Environment for AIX®, Java™ 2 Technology Edition, Version 1.3.0

Licensee agrees that the following terms (in addition to the applicable provisions above) shall apply with respect to any open source provided by International Business Machines Corporation contained within the Product. Notwithstanding anything contained in the CA End User License Agreement, solely with respect to such open source, these terms are not superseded by any written agreement between CA and Licensee:

The IBM Runtime Environment for AIX (R), Java (TM) 2 Technology Edition, Version 1.3.0 (the "Program") is owned by International Business Machines Corporation or one of its subsidiaries (IBM) or an IBM supplier, and is copyrighted and licensed, not sold.

No Warranty

SUBJECT TO ANY STATUTORY WARRANTIES WHICH CAN NOT BE EXCLUDED, IBM MAKES NO WARRANTIES OR CONDITIONS EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, THE WARRANTY OF NON-INFRINGEMENT AND THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, REGARDING THE PROGRAM OR TECHNICAL SUPPORT, IF ANY. IBM MAKES NO WARRANTY REGARDING THE CAPABILITY OF THE PROGRAM TO CORRECTLY PROCESS, PROVIDE AND/OR RECEIVE DATE DATA WITHIN AND BETWEEN THE 20TH AND 21ST CENTURIES. This does not affect any warranties contained in any other applicable agreement between Licensee and CA.

The exclusion also applies to any of IBM's subcontractors, suppliers, or program developers (collectively called "Suppliers").

Limitation of Liability

NEITHER IBM NOR ITS SUPPLIERS WILL BE LIABLE FOR ANY DIRECT OR INDIRECT DAMAGES, INCLUDING WITHOUT LIMITATION, LOST PROFITS, LOST SAVINGS, OR ANY INCIDENTAL, SPECIAL, OR OTHER ECONOMIC CONSEQUENTIAL DAMAGES, EVEN IF IBM IS INFORMED OF THEIR POSSIBILITY. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE ABOVE EXCLUSION OR LIMITATION MAY NOT APPLY TO LICENSEE.

General

Licensee recognizes IBM's and Sun's ownership and title to their respective trademarks and of any goodwill attaching thereto, including goodwill resulting from use. Licensee will not use or attempt to register any trademark, which is confusingly similar to such IBM or Sun trademarks.

Licensee acknowledges that IBM may terminate Licensee's license if Licensee fails to comply with the terms of this agreement. If IBM does so, Licensee must immediately destroy the Program and all copies Licensee made of it.

With respect to any claim by or against IBM relating to the Program, neither Licensee nor IBM will bring a legal action more than two years after the cause of action arose unless otherwise provided by local law without the possibility of contractual waiver or limitation.

The laws of the country in which Licensee acquires the Program govern this agreement, except 1) in Australia, the laws of the State or Territory in which the transaction is performed govern this agreement; 2) in Albania, Armenia, Belarus, Bosnia/Herzegovina, Bulgaria, Croatia, Czech Republic, Georgia, Hungary, Kazakhstan, Kirghizia, Former Yugoslav Republic of Macedonia (FYROM), Moldova, Poland, Romania, Russia, Slovak Republic, Slovenia, Ukraine, and Federal Republic of Yugoslavia, the laws of Austria govern this agreement; 3) in the United Kingdom, all disputes relating to this agreement will be governed by English Law and will be submitted to the exclusive jurisdiction of the English courts; 4) in Canada, the laws in the Province of Ontario govern this agreement; and 5) in the United States and Puerto Rico, and People's Republic of China, the laws of the State of New York govern this agreement.

Country-unique Terms

AUSTRALIA:

No Warranty:

The following paragraph is added to this Section:

Although IBM specifies that there are no warranties, Licensee may have certain rights under the Trade Practices Act 1974 or other legislation and are only limited to the extent permitted by the applicable legislation.

Limitation of Liability:

The following paragraph is added to this Section:

Where IBM is in breach of a condition or warranty implied by the Trade Practices Act 1974, IBM's liability is limited to the repair or replacement of the goods, or the supply of equivalent goods. Where that condition or warranty relates to right to sell, quiet possession or clear title, or the goods are of a kind ordinarily acquired for personal, domestic or household use or consumption, then none of the limitations in this paragraph apply.

GERMANY:

No Warranty:

The following paragraphs are added to this Section:

The minimum warranty period for Programs is six months.

In case a Program is delivered without Specifications, IBM will only warrant that the Program information correctly describes the Program and that the Program can be used according to the Program information. Licensee has to check the usability according to the Program information within the "money-back guaranty" period.

Limitation of Liability:

The following paragraph is added to this Section:

The limitations and exclusions specified in the agreement will not apply to damages caused by IBM with fraud or gross negligence, and for express warranty.

INDIA:

General:

The following replaces the third paragraph of this Section:

If no suit or other legal action is brought, within two years after the cause of action arose, in respect of any claim that either party may have against the other, the rights of the concerned party in respect of such claim will be forfeited and the other party will stand released from its obligations in respect of such claim.

IRELAND:

No Warranty:

The following paragraph is added to this Section:

Except as expressly provided in these terms and conditions, all statutory conditions, including all warranties implied, but without prejudice to the generality of the foregoing, all warranties implied by the Sale of Goods Act 1893 or the Sale of Goods and Supply of Services Act 1980 are hereby excluded.

ITALY:

Limitation of Liability:

This Section is replaced by the following:

Unless otherwise provided by mandatory law, IBM is not liable for any damages which might arise.

NEW ZEALAND:

No Warranty:

The following paragraph is added to this Section:

Although IBM specifies that there are no warranties, Licensee may have certain rights under the Consumer Guarantees Act 1993 or other legislation which cannot be excluded or limited. The Consumer Guarantees Act 1993 will not apply in respect of any goods or services which IBM provides, if Licensee requires the goods and services for the purposes of a business as defined in that Act.

Limitation of Liability:

The following paragraph is added to this Section:

Where Programs are not acquired for the purposes of a business as defined in the Consumer Guarantees Act 1993, the limitations in this Section are subject to the limitations in that Act.

PEOPLE'S REPUBLIC OF CHINA:

Charges:

The following paragraph is added as a new Section:

All banking charges incurred in the People's Republic of China will be borne by Licensee and those incurred outside the People's Republic of China will be borne by IBM.

UNITED KINGDOM:

Limitation of Liability:

The following paragraph is added to this Section at the end of the first paragraph:

The limitation of liability will not apply to any breach of IBM's obligations implied by Section 12 of the Sale of Goods Act 1979 or Section 2 of the Supply of Goods and Services Act 1982.

IBM zSeries Developer Kit for Linux, Java 2 Technology Edition

Terms and Conditions for the Use of IBM® zSeries Developer Kit for Linux®, Java™ 2 Technology Edition

Licensee agrees that the following terms (in addition to the applicable provisions above) shall apply with respect to any open source provided by International Business Machines Corporation contained within the Product. Notwithstanding anything contained in the CA End User License Agreement, solely with respect to such open source, these terms are not superseded by any written agreement between CA and Licensee:

The IBM(R) zSeries Developer Kit for Linux(R), Java(TM) 2 Technology Edition (the "Program") is owned by International Business Machines Corporation or one of its subsidiaries (IBM) or an IBM supplier, and is copyrighted and licensed, not sold.

No Warranty

Licensee acknowledges that the Program is provided 'AS IS'.

SUBJECT TO ANY STATUTORY WARRANTIES WHICH CAN NOT BE EXCLUDED, IBM MAKES NO WARRANTIES OR CONDITIONS EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, THE WARRANTY OF NON-INFRINGEMENT AND THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, REGARDING THE PROGRAM OR TECHNICAL SUPPORT, IF ANY. IBM MAKES NO WARRANTY REGARDING THE CAPABILITY OF THE PROGRAM TO CORRECTLY PROCESS, PROVIDE AND/OR RECEIVE DATE DATA WITHIN AND BETWEEN THE 20TH AND 21ST CENTURIES. This does not affect any warranties contained in any other applicable agreement between Licensee and CA.

The exclusion also applies to any of IBM's subcontractors, suppliers, or program developers (collectively called "Suppliers").

Limitation of Liability

NEITHER IBM NOR ITS SUPPLIERS WILL BE LIABLE FOR ANY DIRECT OR INDIRECT DAMAGES, INCLUDING WITHOUT LIMITATION, LOST PROFITS, LOST SAVINGS, OR ANY INCIDENTAL, SPECIAL, OR OTHER ECONOMIC CONSEQUENTIAL DAMAGES, EVEN IF IBM IS INFORMED OF THEIR POSSIBILITY. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE ABOVE EXCLUSION OR LIMITATION MAY NOT APPLY TO LICENSEE.

General

THIS PROGRAM HAS BEEN PROVIDED TO LICENSEE AT NO CHARGE.

Licensee recognizes IBM's and Sun's ownership and title to their respective trademarks and of any goodwill attaching thereto, including goodwill resulting from use. Licensee will not use or attempt to register any trademark, which is confusingly similar to such IBM or Sun trademarks.

Licensee acknowledges that IBM may terminate Licensee's license if Licensee fails to comply with these terms and conditions. If IBM does so, Licensee must immediately destroy the Program and all copies Licensee made of it.

With respect to any claim by or against IBM relating to the Program, neither Licensee nor IBM will bring a legal action more than two years after the cause of action arose unless otherwise provided by local law without the possibility of contractual waiver or limitation.

The laws of the country in which Licensee acquires the Program govern this agreement, except 1) in Australia, the laws of the State or Territory in which the transaction is performed govern this agreement; 2) in Albania, Armenia, Belarus, Bosnia/Herzegovina, Bulgaria, Croatia, Czech Republic, Georgia, Hungary, Kazakhstan, Kirghizia, Former Yugoslav Republic of Macedonia (FYROM), Moldova, Poland, Romania, Russia, Slovak Republic, Slovenia, Ukraine, and Federal Republic of Yugoslavia, the laws of Austria govern this agreement; 3) in the United Kingdom, all disputes relating to this agreement will be governed by English Law and will be submitted to the exclusive jurisdiction of the English courts; 4) in Canada, the laws in the Province of Ontario govern this agreement; and 5) in the United States and Puerto Rico, and People's Republic of China, the laws of the State of New York govern this agreement.

Country-unique Terms

AUSTRALIA:

No Warranty:

The following paragraph is added to this Section:

Although IBM specifies that there are no warranties, Licensee may have certain rights under the Trade Practices Act 1974 or other legislation and are only limited to the extent permitted by the applicable legislation.

Limitation of Liability:

The following paragraph is added to this Section:

Where IBM is in breach of a condition or warranty implied by the Trade Practices Act 1974, IBM's liability is limited to the repair or replacement of the goods, or the supply of equivalent goods. Where that condition or warranty relates to right to sell, quiet possession or clear title, or the goods are of a kind ordinarily acquired for personal, domestic or household use or consumption, then none of the limitations in this paragraph apply.

GERMANY:

No Warranty:

The following paragraphs are added to this Section:

The minimum warranty period for Programs is six months.

In case a Program is delivered without Specifications, IBM will only warrant that the Program information correctly describes the Program and that the Program can be used according to the Program information. Licensee has to check the usability according to the Program information within the "money-back guaranty" period.

Limitation of Liability:

The following paragraph is added to this Section:

The limitations and exclusions specified in the agreement will not apply to damages caused by IBM with fraud or gross negligence, and for express warranty.

INDIA:

General:

The following replaces the fourth paragraph of this Section:

If no suit or other legal action is brought, within two years after the cause of action arose, in respect of any claim that either party may have against the other, the rights of the concerned party in respect of such claim will be forfeited and the other party will stand released from its obligations in respect of such claim.

IRELAND:

No Warranty:

The following paragraph is added to this Section:

Except as expressly provided in these terms and conditions, all statutory conditions, including all warranties implied, but without prejudice to the generality of the foregoing, all warranties implied by the Sale of Goods Act 1893 or the Sale of Goods and Supply of Services Act 1980 are hereby excluded.

ITALY:

Limitation of Liability:

This Section is replaced by the following:

Unless otherwise provided by mandatory law, IBM is not liable for any damages which might arise.

NEW ZEALAND:

No Warranty:

The following paragraph is added to this Section:

Although IBM specifies that there are no warranties, Licensee may have certain rights under the Consumer Guarantees Act 1993 or other legislation which cannot be excluded or limited. The Consumer Guarantees Act 1993 will not apply in respect of any goods or services which IBM provides, if Licensee requires the goods and services for the purposes of a business as defined in that Act.

Limitation of Liability:

The following paragraph is added to this Section:

Where Programs are not acquired for the purposes of a business as defined in the Consumer Guarantees Act 1993, the limitations in this Section are subject to the limitations in that Act.

PEOPLE'S REPUBLIC OF CHINA:

Charges:

The following paragraph is added as a new Section:

All banking charges incurred in the People's Republic of China will be borne by Licensee and those incurred outside the People's Republic of China will be borne by IBM.

UNITED KINGDOM:

Limitation of Liability:

The following paragraph is added to this Section at the end of the first paragraph:

The limitation of liability will not apply to any breach of IBM's obligations implied by Section 12 of the Sales of Goods Act 1979 or Section 2 of the Supply of Goods and Services Act 1982.

ICU License

ICU License - ICU 1.8.1 and later

COPYRIGHT AND PERMISSION NOTICE

Copyright (c) 1995-2003 International Business Machines Corporation and others

All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

ImageMagick Studio

Copyright© 2003 ImageMagick Studio LLC, a non-profit organization dedicated to making software imaging solutions freely available.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files ("ImageMagick"), to deal in ImageMagick without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of ImageMagick, and to permit persons to whom the ImageMagick is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of ImageMagick.

The software is provided as is, without warranty of any kind, express or implied, including, but not limited to the warranties of merchantability, fitness for a particular purpose and noninfringement. In no event shall ImageMagick Studio LLC be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with ImageMagick or the use or other dealings in ImageMagick.

Except as contained in this notice, the name of the ImageMagick Studio LLC shall not be used in advertising or otherwise to promote the sale, use or other dealings in ImageMagick without prior written authorization from the ImageMagick Studio LLC.

Full Text of Copyright Notices

Copyright© 2002 ImageMagick Studio, a non-profit organization dedicated to making software imaging solutions freely available.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files ("ImageMagick"), to deal in ImageMagick without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of ImageMagick, and to permit persons to whom the ImageMagick is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of ImageMagick.

The software is provided "as is", without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose and noninfringement. In no event shall ImageMagick Studio be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with ImageMagick or the use or other dealings in ImageMagick.

Except as contained in this notice, the name of the ImageMagick Studio shall not be used in advertising or otherwise to promote the sale, use or other dealings in ImageMagick without prior written authorization from the ImageMagick Studio.

Java 2 Runtime Environment (J2RE), Standard Edition, Version 1.4.x

Terms and Conditions for the Use of Java™ 2 Runtime Environment (J2RE), Standard Edition, Version 1.4.x

Licensee agrees that the following terms (in addition to the applicable provisions above) shall apply with respect to any open source provided by Sun Microsystems, Inc. contained within the Product. Notwithstanding anything contained in the CA End User License Agreement ("EULA"), solely with respect to such open source, these terms are not superseded by any written agreement between CA and Licensee:

Title to Java™ 2 Runtime Environment (J2RE), Standard Edition, Version 1.4.x (the "Software") and all associated intellectual property rights are retained by Sun Microsystems, Inc. ("Sun") and/or its licensors. Licensee acknowledges that the Software is not designed or intended for use in the design, construction, operation or maintenance of any nuclear facility. Sun disclaims any express or implied warranty of fitness for such uses. No right, title or interest in or to any trademark, service mark, logo or trade name of Sun or its licensors is granted under this agreement.

The Software is provided "AS IS". As to any claim made by Licensee against Sun respecting the Software, Licensee's exclusive remedy and Sun's entire liability under this limited warranty will be at Sun's option to replace the Software media or refund the fee paid for the Software which Licensee acknowledges is \$0. The foregoing shall not affect any warranties provided in the EULA or any other applicable agreement between Licensee and CA.

UNLESS SPECIFIED IN THIS AGREEMENT, ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A

PARTICULAR PURPOSE OR NON-INFRINGEMENT ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT THESE DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR SPECIAL, INDIRECT, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF OR RELATED TO THE USE OF OR INABILITY TO USE THE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. In no event will Sun's liability exceed the amount paid by Licensee to Sun for the Software under this agreement which Licensee acknowledges is \$0. The foregoing limitations will apply even if the above stated warranty fails of its essential purpose.

Sun may terminate Licensee's right to use the Software if Licensee fails to comply with any provision of this agreement. Upon such termination, Licensee must destroy all copies of the Software.

Notwithstanding anything to the contrary contained in any agreement between Licensee and CA, any action in which Sun is a party will be governed by California law and controlling U.S. federal law. No choice of law rules of any jurisdiction will apply.

Licensee acknowledges that the Software may automatically download, install, and execute applets, applications, software extensions, and updated versions of the Software from Sun ("Software Updates"), which may require Licensee to accept updated terms and conditions for installation. If additional terms and conditions are not presented on installation, the Software Updates will be considered part of the Software and subject to the terms and conditions of the agreement.

Licensee acknowledges that, by Licensee's use of the Software and/or by requesting services that require use of the Software, the Software may automatically download, install, and execute software applications from sources other than Sun ("Other Software"). Sun makes no representations of a relationship of any kind to licensors of Other Software. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR SPECIAL, INDIRECT, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF OR RELATED TO THE USE OF OR INABILITY TO USE OTHER SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Licensee acknowledges and agrees as between Licensee and Sun that Sun owns the SUN, SOLARIS, JAVA, JINI, FORTE, and iPLANET trademarks and all SUN, SOLARIS, JAVA, JINI, FORTE, and iPLANET-related trademarks, service marks, logos and other brand designations ("Sun Marks"), and Licensee agrees to comply with the Sun Trademark and Logo Usage Requirements currently located at <http://www.sun.com/policies/trademarks>. Any use Licensee makes of the Sun Marks inures to Sun's benefit.

Licensee acknowledges that Sun may terminate this agreement immediately should the Software become, or in Sun's opinion be likely to become, the subject of a claim of infringement of any intellectual property right.

For inquiries please contact: Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A. (LFI#119611/Form ID#011801)

Java Naming and Directory Interface (JNDI), Version 1.2.1

Terms and Conditions for the Use of Java Naming and Directory Interface™ (JNDI), Version 1.2.1 and any of the following:

DNS Service Provider Version 1.2

LDAP Service Provider Version 1.2.4

NIS Service Provider Version 1.2.1

RMI Registry Service Provider Version 1.2.1

FS Context Service Provider Version 1.2 beta 3 release

COS Naming Service Provider Version 1.2.1

DSML v1 Service Provider Version 1.2

JNDI/LDAP Booster Pack Version 1.0

Licensee agrees that the following terms (in addition to the applicable provisions above) shall apply with respect to any open source provided by Sun Microsystems, Inc. contained within the Product. Notwithstanding anything contained in the CA End User License Agreement, solely with respect to such open source, these terms are not superseded by any written agreement between CA and Licensee:

Title to Java Naming and Directory Interface™ (JNDI), Version 1.2.1 and the above named Service Providers (collectively the "Software") and all associated intellectual property rights is retained by Sun Microsystems, Inc. ("Sun") and/or its licensors. Licensee acknowledges that the Software is not designed, licensed or intended for use in the design, construction, operation or maintenance of any nuclear facility. Sun disclaims any express or implied warranty of fitness for such uses. No right, title or interest in or to any trademark, service mark, logo or trade name of Sun or its licensors is granted under this agreement.

The Software is provided "AS IS". As to any claim made by Licensee against Sun respecting the Software, Licensee's exclusive remedy and Sun's entire liability under this limited warranty will be at Sun's option to replace the Software media or refund the fee paid by Licensee to Sun for the Software which Licensee acknowledges is \$0. The foregoing shall not affect any warranties provided in any other applicable agreement between Licensee and CA.

UNLESS SPECIFIED IN THIS AGREEMENT, ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT THESE DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR SPECIAL, INDIRECT, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF OR RELATED TO THE USE OF OR INABILITY TO USE THE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. The foregoing limitations will apply even if the above stated warranty fails of its essential purpose.

Sun may terminate Licensee's right to use the Software if Licensee fails to comply with any provision of this agreement. Upon termination, Licensee must destroy all copies of the Software.

Notwithstanding anything to the contrary contained in any agreement between Licensee and CA, any action related to this agreement in which Sun is a party will be governed by California law and controlling U.S. federal law. No choice of law rules of any jurisdiction will apply.

Licensee acknowledges and agrees as between Licensee and Sun that Sun owns the SUN, SOLARIS, JAVA, JINI, FORTE, STAROFFICE, STARPORTAL and iPLANET trademarks and all SUN, SOLARIS, JAVA, JINI, FORTE, STAROFFICE, STARPORTAL and iPLANET-related trademarks, service marks, logos and other brand designations ("Sun Marks"), and Licensee agrees to comply with the Sun Trademark and Logo Usage Requirements currently located at <http://www.sun.com/policies/trademarks>. Any use Licensee makes of the Sun Marks inures to Sun's benefit.

Licensee acknowledges that Sun may terminate this agreement immediately should the Software become, or in Sun's opinion be likely to become, the subject of a claim of infringement of any intellectual property right.

For inquiries please contact: Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A

(LFI#107226/Form ID#011801)

Java XML Pack Summer '02 Bundle

Terms and Conditions for the Use of JAVA™ XML PACK SUMMER '02 BUNDLE

Licensee agrees that the following terms (in addition to the applicable provisions above) shall apply with respect to any open source provided by Sun Microsystems, Inc. contained within the Product. Notwithstanding anything contained in the CA End User License Agreement, solely with respect to such open source, these terms are not superseded by any written agreement between CA and Licensee:

Title to Java™ XML Pack Summer '02 Bundle (the "Software") and all associated intellectual property rights is retained by Sun Microsystems, Inc. ("Sun") and/or its licensors. Licensee acknowledges that the Software is not designed or intended for use in the design, construction, operation or maintenance of any nuclear facility. Sun disclaims any express or implied warranty of fitness for such uses. No right, title or interest in or to any trademark, service mark, logo or trade name of Sun or its licensors is granted under this agreement.

The Software is provided "AS IS". As to any claim made by Licensee against Sun respecting the Software, Licensee's exclusive remedy and Sun's entire liability under this limited warranty will be at Sun's option to replace the Software media or refund the fee paid by Licensee to Sun for the Software which Licensee acknowledges is \$0. The foregoing shall not affect any warranties provided in any other applicable agreement between Licensee and CA.

UNLESS SPECIFIED IN THIS AGREEMENT, ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT THESE DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR SPECIAL, INDIRECT, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF OR RELATED TO THE USE OF OR INABILITY TO USE THE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. The foregoing limitations will apply even if the above stated warranty fails of its essential purpose.

Sun may terminate Licensee's right to use the Software if Licensee fails to comply with any provision of this agreement. Upon termination, Licensee must destroy all copies of the Software.

Notwithstanding anything to the contrary contained in any agreement between Licensee and CA, any action related to this agreement in which Sun is a party will be governed by California law and controlling U.S. federal law. No choice of law rules of any jurisdiction will apply.

Licensee acknowledges and agrees as between Licensee and Sun that Sun owns the SUN, SOLARIS, JAVA, JINI, FORTE, and iPLANET trademarks and all SUN, SOLARIS, JAVA, JINI, FORTE, and iPLANET-related trademarks, service marks, logos and other brand designations ("Sun Marks"), and Licensee agrees to comply with the Sun Trademark and Logo Usage Requirements currently located at <http://www.sun.com/policies/trademarks>. Any use you make of the Sun Marks inures to Sun's benefit.

Licensee acknowledges that Sun may terminate this agreement immediately should the Software become, or in Sun's opinion be likely to become, the subject of a claim of infringement of any intellectual property right.

For inquiries please contact: Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, California 94303

(LFI#113314/Form ID#011801)

JavaBeans Activation Framework, Version 1.0.2

Terms and Conditions for the Use of JavaBeans™ Activation Framework, Version 1.0.2

Licensee agrees that the following terms (in addition to the applicable provisions above) shall apply with respect to any open source provided by Sun Microsystems, Inc. contained within the Product. Notwithstanding anything contained in the CA End User License Agreement, solely with respect to such open source, these terms are not superseded by any written agreement between CA and Licensee:

Title to JavaBeans™ Activation Framework, Version 1.0.2 (the "Software") and all associated intellectual property rights are retained by Sun Microsystems, Inc. ("Sun") and/or its licensors. Licensee acknowledges that the Software is not designed, licensed or intended for use in the design, construction, operation or maintenance of any nuclear facility. Sun disclaims any express or implied warranty of fitness for such uses. No right, title or interest in or to any trademark, service mark, logo or trade name of Sun or its licensors is granted under this agreement.

The Software is provided "AS IS". As to any claim made by Licensee against Sun respecting the Software, Licensee's exclusive remedy and Sun's entire liability under this limited warranty will be at Sun's option to replace the Software media or refund the fee paid by Licensee to Sun for the Software which Licensee acknowledges is \$0. The foregoing shall not affect any warranties provided in any other applicable agreement between Licensee and CA.

UNLESS SPECIFIED IN THIS AGREEMENT, ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT THESE DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR SPECIAL, INDIRECT, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF OR RELATED TO THE USE OF OR INABILITY TO USE THE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. The foregoing limitations will apply even if the above stated warranty fails of its essential purpose.

Sun may terminate Licensee's right to use the Software if Licensee fails to comply with any provision of this agreement. Upon termination, Licensee must destroy all copies of the Software.

Notwithstanding anything to the contrary contained in any agreement between Licensee and CA, any action related to this agreement in which Sun is a party will be governed by California law and controlling U.S. federal law. No choice of law rules of any jurisdiction will apply.

Licensee acknowledges that Sun is under no obligation to support the Software or to provide Licensee with updates or error corrections. Licensee acknowledges that the Software may have defects or deficiencies, which cannot or will not be corrected by Sun.

Licensee acknowledges and agrees as between Licensee and Sun that Sun owns the SUN, SOLARIS, JAVA, JINI, FORTE, and iPLANET trademarks and all SUN, SOLARIS, JAVA, JINI, FORTE, and iPLANET-related trademarks, service marks, logos and other brand designations ("Sun Marks"), and Licensee agrees to comply with the Sun Trademark and Logo Usage Requirements currently located at <http://www.sun.com/policies/trademarks>. Any use Licensee makes of the Sun Marks inures to Sun's benefit.

Licensee acknowledges that Sun may terminate this agreement immediately should the Software become, or in Sun's opinion be likely to become, the subject of a claim of infringement of any intellectual property right.

For inquiries please contact: Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, California 94303

(LFI#115020/Form ID#011801)

JavaMail, Version 1.3

Terms and Conditions for the Use of JavaMail™, Version 1.3

Licensee agrees that the following terms (in addition to the applicable provisions above) shall apply with respect to any open source provided by Sun Microsystems, Inc. contained within the Product. Notwithstanding anything contained in the CA End User License Agreement, solely with respect to such open source, these terms are not superseded by any written agreement between CA and Licensee:

Title to JavaMail™, Version 1.3 (the "Software") and all associated intellectual property rights is retained by Sun Microsystems, Inc. ("Sun") and/or its licensors. Licensee acknowledges that the Software is not designed, licensed or intended for use in the design, construction, operation or maintenance of any nuclear facility. Sun disclaims any express or implied warranty of fitness for such uses. No right, title or interest in or to any trademark, service mark, logo or trade name of Sun or its licensors is granted under this agreement.

The Software is provided "AS IS". As to any claim made by Licensee against Sun respecting the Software, Licensee's exclusive remedy and Sun's entire liability under this limited warranty will be at Sun's option to replace the Software media or refund the fee paid by Licensee to Sun for the Software which Licensee acknowledges is \$0. The foregoing shall not affect any warranties provided in any other applicable agreement between Licensee and CA.

UNLESS SPECIFIED IN THIS AGREEMENT, ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT THESE DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR SPECIAL, INDIRECT, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF OR RELATED TO THE USE OF OR INABILITY TO USE THE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. The foregoing limitations will apply even if the above stated warranty fails of its essential purpose.

Sun may terminate Licensee's right to use the Software if Licensee fails to comply with any provision of this agreement. Upon termination, Licensee must destroy all copies of the Software.

Notwithstanding anything to the contrary contained in any agreement between Licensee and CA, any action related to this agreement in which Sun is a party will be governed by California law and controlling U.S. federal law. No choice of law rules of any jurisdiction will apply.

Licensee acknowledges and agrees as between Licensee and Sun that Sun owns the SUN, SOLARIS, JAVA, JINI, FORTE, STAROFFICE, STARPORTAL and iPLANET trademarks and all SUN, SOLARIS, JAVA, JINI, FORTE, STAROFFICE, STARPORTAL and iPLANET-related trademarks, service marks, logos and other brand designations ("Sun Marks"), and Licensee agrees to comply with the Sun Trademark and Logo Usage Requirements currently located at <http://www.sun.com/policies/trademarks>. Any use Licensee makes of the Sun Marks inures to Sun's benefit.

Licensee acknowledges that Sun may terminate this agreement immediately should the Software become, or in Sun's opinion be likely to become, the subject of a claim of infringement of any intellectual property right.

For inquiries please contact: Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A

(LFI#114176/Form ID#011801)

Sun Microsystems, Inc. Java 2 Runtime Environment 1.4.2

Terms and Conditions for the Use of Java™ 2 Runtime Environment (J2RE), Standard Edition, Version 1.4.2_X

Licensee agrees that the following terms (in addition to the applicable provisions above) shall apply with respect to any open source provided by Sun Microsystems, Inc. contained within the Product. Notwithstanding anything contained in the CA End User License Agreement, solely with respect to such open source, these terms are not superseded by any written agreement between CA and Licensee:

"Software" means Java™ 2 Runtime Environment (J2RE), Standard Edition, Version 1.4.2_X and any user manuals, programming guides and other documentation provided to Licensee.

Title to Software and all associated intellectual property rights are retained by Sun Microsystems, Inc. ("Sun") and/or its licensors. Licensee acknowledges that Software is not designed or intended for use in the design, construction, operation or maintenance of any nuclear facility. Sun disclaims any express or implied warranty of fitness for such uses. No right, title or interest in or to any trademark, service mark, logo or trade name of Sun or its licensors is granted under this agreement.

The Software is provided "AS IS". As to any claim made by Licensee against Sun respecting Software, Licensee's exclusive remedy and Sun's entire liability under this limited warranty will be at Sun's option to replace Software media or refund the fee paid for Software by Licensee to Sun which Licensee acknowledges is \$0.

UNLESS SPECIFIED IN THIS AGREEMENT, ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT THESE DISCLAIMERS ARE HELD TO BE LEGALLY INVALID. The foregoing limitations shall not affect any warranties provided in any other applicable agreement between Licensee and CA.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR SPECIAL, INDIRECT, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF OR RELATED TO THE USE OF OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. The foregoing limitations will apply even if the above stated warranty fails of its essential purpose.

Licensee acknowledges that at Licensee's request or consent optional features of the Software may download, install, and execute applets, applications, software extensions, and updated versions of the Software from Sun ("Software Updates"), which may require Licensee to accept updated terms and conditions for installation. If additional terms and conditions are not presented on installation, the Software Updates will be considered part of the Software and subject to the terms and conditions of this agreement.

Licensee acknowledges that, by Licensee's use of optional features of the Software and/or by requesting services that require use of the optional features of the Software, the Software may automatically download, install, and execute software applications from sources other than Sun ("Other Software"). Sun makes no representations of a relationship of any kind to licensors of Other Software. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR SPECIAL, INDIRECT, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF OR RELATED TO THE USE OF OR INABILITY TO USE OTHER SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Some states do not allow the exclusion of incidental or consequential damages, so some of the terms above may not be applicable to Licensee.

Licensee acknowledges that Sun may terminate Licensee's use of the Software without notice if Licensee fails to comply with any provision of this agreement. Licensee acknowledges that Sun may terminate this agreement immediately should the Software become, or in Sun's opinion be likely to become, the subject of a claim of infringement of any intellectual property right. Upon termination, Licensee must destroy all copies of Software.

Licensee acknowledges and agrees as between Licensee and Sun that Sun owns the SUN, SOLARIS, JAVA, JINI, FORTE, and iPLANET trademarks and all SUN, SOLARIS, JAVA, JINI, FORTE, and iPLANET-related trademarks, service marks, logos and other brand designations ("Sun Marks"), and Licensee agrees to comply with the Sun Trademark and Logo Usage Requirements currently located at <http://www.sun.com/policies/trademarks>. Any use Licensee makes of the Sun Marks inures to Sun's benefit.

Notwithstanding anything to the contrary contained in any agreement between Licensee and CA, any action related to this agreement in which Sun is a party will be governed by California law and controlling U.S. federal law. No choice of law rules of any jurisdiction will apply.

Licensee acknowledges that additional copyright notices and license terms applicable to portions of the Software are set forth in the THIRDPARTYLICENSEREADME.txt file.

For inquiries please contact: Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A.

(LFI#135002/Form ID#011801)

JUnit 3.8.1

This product includes junit 3.8.1 from (www.junit.org)

JUnit is a regression testing framework written by Erich Gamma and Kent Beck. It is used by the developer who implements unit tests in Java. JUnit is Open Source Software, released under the IBM's Common Public License Version 1.0 and hosted on SourceForge.

Common Public License Version 1.0

THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS COMMON PUBLIC LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT'S ACCEPTANCE OF THIS AGREEMENT.

1. DEFINITIONS

"Contribution" means:

a) in the case of the initial Contributor, the initial code and documentation distributed under this Agreement, and

b) in the case of each subsequent Contributor:

i) changes to the Program, and

ii) additions to the Program;

where such changes and/or additions to the Program originate from and are distributed by that particular Contributor. A Contribution 'originates' from a Contributor if it was added to the Program by such Contributor itself or anyone acting on such Contributor's behalf. Contributions do not include additions to the Program which: (i) are separate modules of software distributed in conjunction with the Program under their own license agreement, and (ii) are not derivative works of the Program.

"Contributor" means any person or entity that distributes the Program.

"Licensed Patents " mean patent claims licensable by a Contributor which are necessarily infringed by the use or sale of its Contribution alone or when combined with the Program.

"Program" means the Contributions distributed in accordance with this Agreement.

"Recipient" means anyone who receives the Program under this Agreement, including all Contributors.

2. GRANT OF RIGHTS

a) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free copyright license to reproduce, prepare derivative works of, publicly display, publicly perform, distribute and sublicense the Contribution of such Contributor, if any, and such derivative works, in source code and object code form.

b) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free patent license under Licensed Patents to make, use, sell, offer to sell, import and otherwise transfer the Contribution of such Contributor, if any, in source code and object code form. This patent license shall apply to the combination of the Contribution and the Program if, at the time the Contribution is added by the Contributor, such addition of the Contribution causes such combination to be covered by the Licensed Patents. The patent license shall not apply to any other combinations which include the Contribution. No hardware per se is licensed hereunder.

c) Recipient understands that although each Contributor grants the licenses to its Contributions set forth herein, no assurances are provided by any Contributor that the Program does not infringe the patent or other intellectual property rights of any other entity. Each Contributor disclaims any liability to Recipient for claims brought by any other entity based on infringement of intellectual property rights or otherwise. As a condition to exercising the rights and licenses granted hereunder, each Recipient hereby assumes sole responsibility to secure any other intellectual property rights needed, if any. For example, if a third party patent license is required to allow Recipient to distribute the Program, it is Recipient's responsibility to acquire that license before distributing the Program.

d) Each Contributor represents that to its knowledge it has sufficient copyright rights in its Contribution, if any, to grant the copyright license set forth in this Agreement.

3. REQUIREMENTS

A Contributor may choose to distribute the Program in object code form under its own license agreement, provided that:

a) it complies with the terms and conditions of this Agreement; and

b) its license agreement:

i) effectively disclaims on behalf of all Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose;

ii) effectively excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits;

iii) states that any provisions which differ from this Agreement are offered by that Contributor alone and not by any other party; and iv) states that source code for the Program is available from such Contributor, and informs licensees how to obtain it in a reasonable manner on or through a medium customarily used for software exchange.

When the Program is made available in source code form:

a) it must be made available under this Agreement; and

b) a copy of this Agreement must be included with each copy of the Program.

Contributors may not remove or alter any copyright notices contained within the Program. Each Contributor must identify itself as the originator of its Contribution, if any, in a manner that reasonably allows subsequent Recipients to identify the originator of the Contribution.

4. COMMERCIAL DISTRIBUTION

Commercial distributors of software may accept certain responsibilities with respect to end users, business partners and the like. While this license is intended to facilitate the commercial use of the Program, the Contributor who includes the Program in a commercial product offering should do so in a manner which does not create potential liability for other Contributors. Therefore, if a Contributor includes the Program in a commercial product offering, such Contributor ("Commercial Contributor") hereby agrees to defend and indemnify every other Contributor ("Indemnified Contributor") against any losses, damages and costs (collectively "Losses") arising from claims, lawsuits and other legal actions brought by a third party against the Indemnified Contributor to the extent caused by the acts or omissions of such Commercial Contributor in connection with its distribution of the Program in a commercial product offering. The obligations in this section do not apply to any claims or Losses relating to any actual or alleged intellectual property infringement. In order to qualify, an Indemnified Contributor must: a) promptly notify the Commercial Contributor in writing of such claim, and b) allow the Commercial Contributor to control, and cooperate with the Commercial Contributor in, the defense and any related settlement negotiations. The Indemnified Contributor may participate in any such claim at its own expense. For example, a Contributor might include the Program in a commercial product offering, Product X. That Contributor is then a Commercial Contributor. If that Commercial Contributor then makes performance claims, or offers warranties related to Product X, those performance claims and warranties are such Commercial Contributor's responsibility alone. Under this section, the Commercial Contributor would have to defend claims against the other Contributors related to those performance claims and warranties, and if a court requires any other Contributor to pay any damages as a result, the Commercial Contributor must pay those damages.

5. NO WARRANTY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NONINFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely responsible for determining the appropriateness of using and distributing the Program and assumes all risks associated with its exercise of rights under this Agreement, including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations.

6. DISCLAIMER OF LIABILITY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. GENERAL

If any provision of this Agreement is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this Agreement, and without further action by the parties hereto, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable. If Recipient institutes patent litigation against a Contributor with respect to a patent applicable to software (including a cross-claim or counterclaim in a lawsuit), then any patent licenses granted by that Contributor to such Recipient under this Agreement shall terminate as of the date such litigation is filed. In addition, if Recipient institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Program itself (excluding combinations of the Program with other software or hardware) infringes such Recipient's patent(s), then such Recipient's rights granted under Section 2(b) shall terminate as of the date such litigation is filed. All Recipient's rights under this Agreement shall terminate if it fails to comply with any of the material terms or conditions of this Agreement and does not cure such failure in a reasonable period of time after becoming aware of such noncompliance. If all Recipient's rights under this Agreement terminate, Recipient agrees to cease use and distribution of the Program as soon as reasonably practicable. However, Recipient's obligations under this Agreement and any licenses granted by Recipient relating to the Program shall continue and survive.

Everyone is permitted to copy and distribute copies of this Agreement, but in order to avoid inconsistency the Agreement is copyrighted and may only be modified in the following manner. The Agreement Steward reserves the right to publish new versions (including revisions) of this Agreement from time to time. No one other than the Agreement Steward has the right to modify this Agreement. IBM is the initial Agreement Steward. IBM may assign the responsibility to serve as the Agreement Steward to a suitable separate entity. Each new version of the Agreement will be given a distinguishing version number. The Program (including Contributions) may always be distributed subject to the version of the Agreement under which it was received. In addition, after a new version of the Agreement is published, Contributor may elect to distribute the Program (including its Contributions) under the new version. Except as expressly stated in Sections 2(a) and 2(b) above, Recipient receives no rights or licenses to the intellectual property of any Contributor under this Agreement, whether expressly, by implication, estoppel or otherwise. All rights in the Program not expressly granted under this Agreement are reserved.

This Agreement is governed by the laws of the State of New York and the intellectual property laws of the United States of America. No party to this Agreement will bring a legal action under this Agreement more than one year after the cause of action arose. Each party waives its rights to a jury trial in any resulting litigation.

OpenSSL 0.9.7c

This product includes OpenSSL 0.9.7c (<http://www.openssl.org/>)

The OpenSSL toolkit stays under a dual license, i.e. both the conditions of the OpenSSL License and the original SSLeay license apply to the toolkit. See below for the actual license texts. Actually both licenses are BSD-style Open Source licenses. In case of any license issues related to OpenSSL please contact openssl-core@openssl.org.

OpenSSL License

Copyright (c) 1998-2001 The OpenSSL Project. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgment:
"This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (<http://www.openssl.org/>)"
4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact openssl-core@openssl.org.
5. Products derived from this software may not be called "OpenSSL" nor may "OpenSSL" appear in their names without prior written permission of the OpenSSL Project.
6. Redistributions of any form whatsoever must retain the following acknowledgment:
"This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (<http://www.openssl.org/>)"

THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT `AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE OpenSSL PROJECT OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This product includes cryptographic software written by Eric Young (eay@cryptsoft.com). This product includes software written by Tim Hudson (tjh@cryptsoft.com).

Original SSLeay License

Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com). All rights reserved.

This package is an SSL implementation written by Eric Young (eay@cryptsoft.com). The implementation was written so as to conform with Netscapes SSL.

This library is free for commercial and non-commercial use as long as the following conditions are adhered to. The following conditions apply to all code found in this distribution, be it the RC4, RSA, lhash, DES, etc., code; not just the SSL code. The SSL documentation included with this distribution is covered by the same copyright terms except that the holder is Tim Hudson (tjh@cryptsoft.com).

Copyright remains Eric Young's, and as such any Copyright notices in the code are not to be removed.

If this package is used in a product, Eric Young should be given attribution as the author of the parts of the library used.

This can be in the form of a textual message at program startup or in documentation (online or textual) provided with the package.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. All advertising materials mentioning features or use of this software must display the following acknowledgement:

"This product includes cryptographic software written by Eric Young (eay@cryptsoft.com)"

The word 'cryptographic' can be left out if the routines from the library being used are not cryptographic related :-).

4. If you include any Windows specific code (or a derivative thereof) from the apps directory (application code) you must include an acknowledgement:

"This product includes software written by Tim Hudson (tjh@cryptsoft.com)"

THIS SOFTWARE IS PROVIDED BY ERIC YOUNG `AS IS' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The licence and distribution terms for any publically available version or derivative of this code cannot be changed. i.e. this code cannot simply be copied and put under another distribution licence [including the GNU Public Licence.]

Sun Microsystems, Inc. JIMI SDK, Version 2.0

Terms and Conditions for the Use of JIMI SDK, Version 2.0

Licensee agrees that the following terms (in addition to the applicable provisions above) shall apply with respect to any open source provided by Sun Microsystems, Inc. contained within the Product. Notwithstanding anything contained in the CA End User License Agreement ("EULA"), solely with respect to such open source, these terms are not superseded by any written agreement between CA and Licensee:

Title to JIMI SDK, Version 2.0 (the "Software") and all associated intellectual property rights are retained by Sun Microsystems, Inc. ("Sun") and/or its licensors.

The Software is provided "AS IS". As to any claim made by Licensee against Sun respecting the Software, licensee's exclusive remedy and Sun's entire liability under this limited warranty will be at Sun's option to replace the Software media or refund the fee paid for the Software which Licensee acknowledges is \$0. The foregoing shall not affect any warranties provided in the EULA or any other applicable agreement between Licensee and CA.

UNLESS SPECIFIED IN THIS AGREEMENT, ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT THESE DISCLAIMERS ARE HELD TO BE LEGALLY INVALID. LICENSEE ACKNOWLEDGES THAT THE SOFTWARE IS NOT DESIGNED OR INTENDED FOR USE IN THE DESIGN, CONSTRUCTION, OPERATION, OR MAINTENANCE OF ANY NUCLEAR FACILITY. SUN DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY OF FITNESS FOR SUCH USES.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR SPECIAL, INDIRECT, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF OR RELATED TO THE USE OF OR INABILITY TO USE THE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. In no event will Sun's liability to Licensee exceed the amount paid by Licensee to Sun for the Software under this agreement which Licensee acknowledges is \$0. The foregoing limitations will apply even if the above stated warranty fails of its essential purpose.

Sun may terminate Licensee's right to use the Software if Licensee fails to comply with any provision of this agreement. Upon such termination, Licensee must destroy all copies of the Software.

Notwithstanding anything to the contrary contained in any agreement between Licensee and CA, any action in which Sun is a party will be governed by California law and controlling U.S. federal law. No choice of law rules of any jurisdiction will apply.

For inquiries please contact: Sun Microsystems, Inc. 901 San Antonio Road, Palo Alto, California 94303.

Is a condition precedent to each license grant in this agreement, Licensee agrees to indemnify, hold harmless, and defend Sun and its licensors from and against any and all claims, lawsuits, liabilities, demands and expenses (including attorneys' fees), that arise or result from the use or distribution of the Software or the Product, including without limitation, those brought by Unisys Corporation, its successors and assigns, with respect to U.S. Patent Number 4,558,302 and all foreign counterparts thereto which Unisys Corporation may now have or acquire in the future (the "LZW Patents") relating to Licensee's making, using, selling, licensing, importing, offering to sell, or otherwise transferring the GIF encoding and/or decoding feature of the Software or the Product. This agreement does not grant any rights to Licensee with respect to the LZW Patents.

Licensee acknowledges that this agreement does not authorize Licensee to use any Sun name, trademark or logo. Licensee acknowledges and agrees as between Licensee and Sun that Sun owns the Java trademark and all Java-related trademarks, logos and icons including the Coffee Cup and Duke ("Java Marks") and Licensee agrees to comply with the Java Trademark Guidelines at <http://java.sun.com/trademarks.html>.

TPSR P05273_11

Details of the TPSR P05273_1

1. TPSR Information

TPSR ID: P05273_1

Project 360 ID: P05273

Project Name: INF-eHealth/Voice Integration Phase 1 PES - Dev (Sub)

Created By: rosjo12

PGC Approver: MOTLU01

TPSR Status: Approved

License Text: COPYRIGHT AND PERMISSION NOTICE

Copyright (c) 1996 - 2008, Daniel Stenberg, <daniel@haxx.se>.

All rights reserved.

Permission to use, copy, modify, and distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright

notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR

IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,

FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD
PARTY RIGHTS. IN

NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
CLAIM,

DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT
OR

OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
SOFTWARE OR THE USE

OR OTHER DEALINGS IN THE SOFTWARE.

Except as contained in this notice, the name of a copyright holder shall not
be used in advertising or otherwise to promote the sale, use or other dealings
in this Software without prior written authorization of the copyright holder.

License URL: <http://curl.haxx.se/docs/copyright.html>

Copyright Text:

```

/*****
*****

```

```

*
*           _ _ _ _ _
* Project   _|||_|_|||
*
*         / _|||_|_|||
*
*       |(_||_|_|_<|_|_

```


Distribution Type: P05273

Localization: P05273

Platforms: Solaris 2.10 , Windows 2003 32-bit,

Creation Date: 2008-12-03

Modification Date: 2008-12-08

2. Component Information

Component: libcurl 7.19.2

Vendor: Daniel Stenberg

Description: Free and easy-to-use client-side URL transfer library. Libcurl is a library for transferring data with URL syntax, supporting FTP, FTPS, HTTP, HTTPS, TFTP, GOPHER, TELNET, DICT, FILE and LDAP. The tool and library offer a myriad of powerful features and full protocol control.

Features: Provides all HTTP/HTTPS functionality to collect data from Cisco CallManager 3.3 - 6.1. Recompiled with CA etpki version 3.0.0 (as used by eHealth 6.1) to be fips compliant. Also provides ipv6 support for cisco data collection. This version also fixes a memory leak when ssl (https) is used that existed in version 7.18.2

Cost: 0

URL: <http://curl.haxx.se/libcurl/>,
<http://sourceforge.net/projects/curl/>

Recommendation: Allowed

CA Version ID: 1

3. Administrator Comments

TAC Admin PMF: albda04

TAC Remarks: None

TAC Action Date: 2008-12-08

4. Legal Comments

Legal PMF: macgl03

Legal Remarks: Approved by WLD on 12/7/08 subject to the following:

1. Permission to use, copy, modify, and distribute this software is permitted, provided that the copyright notice, disclaimer and permission notice set forth in the License Text section of this TPSR appears in all copies.

2. The name of the copyright holder may not be used to endorse or promote products derived from this software without specific prior written permission.

Legal Action Date: 2008-12-07

Installation Requirements: None

Copyright Requirements: None

Documentation Requirements: The entire ?License Text? section must be reproduced in an Acknowledgments section in the CA product user documentation in accordance with the Tech Pub guidelines. Please precede the text of the license agreement with: "This product includes libcurl 7.19.2, the use of which is governed by the following terms:"

Werken digital SAXPath 1.0

This product includes software developed by the SAXPath Project (<http://www.saxpath.org/>). The SAXPath software is distributed in accordance with the following license agreement.

Full Text of the License Agreement:

\$Id: LICENSE,v 1.1 2002/04/26 17:43:56 jstrachan Exp \$

Copyright (C) 2000-2002 werken digital. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions, and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions, and the disclaimer that follows these conditions in the documentation and/or other materials provided with the distribution.
3. The name "SAXPath" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact license@saxpath.org.
4. Products derived from this software may not be called "SAXPath", nor may "SAXPath" appear in their name, without prior written permission from the SAXPath Project Management (pm@saxpath.org).

In addition, we request (but do not require) that you include in the end-user documentation provided with the redistribution and/or in the software itself an acknowledgement equivalent to the following: "This product includes software developed by the SAXPath Project (<http://www.saxpath.org/>)."

Alternatively, the acknowledgment may be graphical using the logos available at <http://www.saxpath.org/>.

THIS SOFTWARE IS PROVIDED ``AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE SAXPath AUTHORS OR THE PROJECT CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This software consists of voluntary contributions made by many individuals on behalf of the SAXPath Project and was originally created by bob mcwhirter <bob@werken.com> and James Strachan <jstrachan@apache.org>. For more information on the SAXPath Project, please see <<http://www.saxpath.org/>>.

Werken Company Jaxen 1.0

This product includes software developed by The Werken Company (<http://www.jaxen.werken.com/>). The Jaxen software is distributed in accordance with the following license agreement.

Full Text of the License Agreement:

\$Id: LICENSE.txt,v 1.3 2003/06/29 18:22:02 ssanders Exp \$

Copyright 2003 (C) The Werken Company. All Rights Reserved.

Redistribution and use of this software and associated documentation ("Software"), with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain copyright statements and notices. Redistributions must also contain a copy of this document.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The name "jaxen" must not be used to endorse or promote products derived from this Software without prior written permission of The Werken Company. For written permission, please contact bob@werken.com.
4. Products derived from this Software may not be called "jaxen" nor may "jaxen" appear in their names without prior written permission of The Werken Company. "jaxen" is a registered trademark of The Werken Company.
5. Due credit should be given to The Werken Company. (<http://jaxen.werken.com/>).

THIS SOFTWARE IS PROVIDED BY THE WERKEN COMPANY AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE WERKEN COMPANY OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Full Text of Copyright Notices:

Copyright© 2000-2002 bob mcwhirter & James Strachan.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions, and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions, and the disclaimer that follows these conditions in the documentation and/or other materials provided with the distribution.
3. The name "Jaxen" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact license@jaxen.org.
4. Products derived from this software may not be called "Jaxen", nor may "Jaxen" appear in their name, without prior written permission from the Jaxen Project Management (pm@jaxen.org).

In addition, we request (but do not require) that you include in the end-user documentation provided with the redistribution and/or in the software itself an acknowledgement equivalent to the following: "This product includes software developed by the Jaxen Project (<http://www.jaxen.org/>)."

Alternatively, the acknowledgment may be graphical using the logos available at <http://www.jaxen.org/>

THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE Jaxen AUTHORS OR THE PROJECT CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This software consists of voluntary contributions made by many individuals on behalf of the Jaxen Project and was originally created by bob mcwhirter <bob@werken.com> and James Strachan <jstrachan@apache.org>. For more information on the Jaxen Project, please see <<http://www.jaxen.org/>>.

Index

.NET • 351
 programming • 357

A

access types • 61
ActiveX controls
 defined • 159
add
 markers • 235
Aion BRE language • 22
Aion terms • 62
aionsession.jar file • 369
Apache Axis • 401
 validation • 402
application libraries • 44
applications • 43
 backing up • 81
 build • 443
 creating • 79
 deploy • 446
 develop application for non-windows • 84
 edit from remote source • 84
 open • 80
 read-only • 81
 restore open • 83
 restoring • 82
 saving • 81
 search across • 137
 standalone • 45
 view • 91
Association Editor • 104
associations
 edit • 104
Attribute Editor • 105
attributes • 47, 63
 Watched Attribute pane • 420
 watchpoints
set and remove • 432
automatic
 data loading • 238
automation
 using Aion as a client • 394

B

back up applications • 81
backward chaining
 debugging • 437
breakpoints • 429
 remove • 431
 set • 433
business rules • 299

C

C and C++ components • 335
 data types • 341
C/C++ terms • 62
change management • 130
 functions • 130
changes
 management • 130
Class Editor • 106
class relationships
 inheritance hierarchy • 49
class relationships,containment • 52
classes • 47, 63
 attributes • 63
 constants • 69
 editing • 106
 methods • 67
clear
 watchpoints • 432
client
 use Aion as automation • 394
client-server applications • 21
COM
 objects
 generation • 392, 394
COM components
 data types • 395
complex data types • 400
component-based development • 21
components
 as Web services • 399
 C and C++ • 335
 Managed C++ • 351
concurrent development • 129
connections

-
- defining for databases • 222
 - Console-Mode Installation • 31
 - constants • 69
 - containment) • 52
 - control groups
 - work with control groups • 156
 - controls
 - ActiveX • 159
 - add to windows • 151
 - create • 154
 - editing • 155
 - logic • 182
 - order • 176
 - properties • 161
 - splitter windows • 160
 - conventions
 - menu • 171
 - toolbar • 175
 - Custom Installation • 29
 - D**
 - data
 - automatic
 - loading mode • 238
 - commit modes • 244
 - load
 - from a database • 237
 - manual
 - commit mode • 245
 - load mode • 239
 - data objects
 - MQLib • 255
 - Data Test facility • 236, 237
 - data types
 - C and C++ • 341
 - COM components • 395
 - complex • 400
 - Managed C++ • 362
 - database support • 23
 - database terms • 62
 - databases
 - define a connection • 222
 - handling errors • 248
 - saving modifications • 244
 - DCOM
 - test configuration • 394
 - Debugger
 - instance counter • 421
 - Method Body pane • 421
 - set watchpoints • 432
 - setting code breakpoints • 431
 - settings • 432
 - Watched Attribute pane • 420
 - debugging
 - embedded • 416
 - how to • 433
 - rule-based inferencing • 436
 - Decision Table Editor • 106
 - deployment • 399
 - as Web services • 399
 - WebLogic • 405
 - WebSphere • 406
 - derived classes • 49
 - development
 - concurrent • 129
 - development environment
 - customizing • 89
 - dialog boxes
 - applications
 - using in • 146
 - edit • 149
 - logic • 179
 - properties • 150
 - dialogs
 - create at runtime • 208
 - domain
 - defined • 286
 - domain interface
 - defined • 289
 - domain interface (DI) members
 - defined • 286
 - Dynamic Rule Manager • 293
 - Dynamic Rulebase Administrator • 293
 - dynamic rules • 291
 - Aion BRE-supplied libraries • 294
 - domain • 286
 - domain interface • 286, 289
 - domain interface members • 286
 - DynRDLib • 293, 294
 - DynRELib • 294
 - DynRLib • 294
 - examples • 294
 - rulebase • 293
 - uses • 292
 - dynamic versus static • 70
 - DynRDLib • 293, 294
 - DynRELib • 294
 - DynRLib • 294
-

E

editors

- Association • 104
 - Attribute • 105
 - Class • 106
 - Decision Table • 106
 - descriptions of • 102
 - Instance • 107
 - Menu • 107
 - Method • 107
 - Query • 109
 - Rule • 108
 - standard
- procedures • 103
- standard tab pages • 103
 - Stored Procedure • 110
 - Tool • 111
 - use editors • 102
 - Window • 111
- embedded debugging • 416
- Entry Class property • 45
- Explorer
- set explorer workspace options • 100
 - use explore • 99
- exported classes
- coding • 355

F

focus

- keyboard • 156
- forward chaining
- debugging • 437

G

graphical user interface(GUI) • 22

Graphical-Mode Installation • 32

graphics

- add to windows • 175
 - layering • 176
 - logic • 185
 - properties • 177
- grouping radio buttons • 156

GUI

- add controls • 151
- add menus • 163
- adding toolbars • 171
- create dialog boxes • 146
- creating windows • 146

implement logic • 178

H

has-a relationship • 52

hierarchy

- class • 49
- how to debug • 433

I

icons

- table • 92

IDE

- icons • 92
- included libraries • 44, 123
- inference
- debugging • 436
- inheritance • 49
- queries • 223
- Install on Linux/UNIX server • 31
- Install on Microsoft Windows • 26
- Installation • 25
- Installation Prerequisites • 25
- instance counter
- Debugger • 421
- Instance Editor • 107
- instances • 47
- dynamic versus static • 70
- interface layers • 21
- C and C++ • 335
 - Java • 363
 - Managed C++ • 351
 - use interface layer • 444
- interfaces • 72
- is-a relationships • 49

J

jar files

- aionsession • 369
- Java components • 363

K

keyboard focus

- ordering • 156

L

layering graphics • 176

libraries • 44

- benefits of • 45

- boundaries of • 46
- for dynamic rules • 294
- included • 123
- properties • 442

load

- data from a database • 237
- manual • 239

logic

- application • 178
- control • 182
- dialog box • 179
- graphic • 185
- implement • 178
- menu • 183
- toolbar • 185
- window • 179

M

- Main class • 45
- Managed C++ components • 351
 - data types • 362
- managed code • 351
- manual commit mode • 245
- markers
 - add • 235
 - use markers with query and classes • 227
- MDI windows
 - defined • 148
- MDI windows, • 148
- members
 - of class • 47
- Menu Editor • 107
- menu items
 - add to menu titles • 165
- menu titles
 - add menu items • 165
 - attach to windows • 166
 - create • 164
- menus
 - add to windows • 163
 - conventions • 171
 - logic • 183
 - pop-up • 166
 - properties • 167
- messages • 47
- Method Body pane • 421
- Method Editor • 22, 107
- Method Editor Options • 108
- method language • 22

- methods • 47, 67
 - accessor • 286
 - application-defined • 223
 - create • 190
 - editing • 107
 - WhenFetched() method • 241
 - WhenUpdated() method • 246
- modes
 - automatic commit • 245
 - manual commit • 245
- MQLib • 253
 - data objects • 255
- MQSeries • 253

N

- new method • 190

O

- object communication • 21
 - interface layers • 21
- object orientation • 20
 - inheritance • 49
 - overview • 47
 - polymorphism • 50
 - terminology • 47
- object orientation,containment • 52
- objects • 47
 - COM
- generate • 392
 - copy • 117
 - delete • 118
 - icons representing • 92
 - pasting • 117
 - properties of GUI • 141
 - search for • 136
- open applications
 - restore • 83
- options
 - Method Editor • 108
- order
 - graphics • 176
- Output Window • 93

P

- panes
 - Debugger
 - Method Body • 421
 - Watched Attribute • 420
- parser utility • 232

- SQL • 232
- Paster utility
 - SQL • 232
- pattern matching
 - debugging • 437
- polymorphism • 50
- pop-up menus • 166
- private access type • 61
- Project Workspace • 96
- properties
 - dialog box • 150
 - graphics • 177
 - GUI objects • 141
 - menu • 167
 - toolbar • 173
 - window • 150
- properties, • 161
- protected access type • 61
- public access type • 61

Q

- queries
 - create • 224
 - define • 222
 - inheritance • 223
 - modify • 227
 - reuse • 223
 - stored procedures • 233
- queries, • 223
- Query Editor • 109
- Queue Manager • 254

R

- radio buttons
 - grouping • 156
- Rapid Application Development (RAD) • 23
- read-only applications • 81
- remote source
 - develop applications for non-windows • 84
 - editing applications from • 84
- removing
 - watchpoints • 432
- resources
 - create • 177
- reuse queries • 223
- Rule Editor • 102, 108
- Rule Manager Wizard • 299
- rules
 - business rules • 299

- decision table • 291
- inferencing, • 19
- static • 291

S

- safeguards, source control • 127
- saving
 - database modifications • 244
- SDI windows • 148
- set
 - breakpoints • 433
 - watchpoints • 432
- settings
 - Debugger • 432
- Setup • 25
- Silent Installation • 40
- single class hierarchy • 43
- source control • 126
 - program options • 128
 - safeguards • 127
- source files
 - restore applications from • 82
- specialization • 49
- splitter windows • 160
- SQL
 - Paster utility • 232
 - write • 231
- SQL Paster utility • 232
- Start method • 45
- static versus dynamic • 70
- Stored Procedure Editor • 110
- stored procedures • 110
 - create • 233
 - define • 232
 - editor • 233
 - queries • 233
- subclasses • 49

T

- tab controls
 - tab controls • 158
- tab pages • 94
 - Output Window • 94
 - standard in editors • 103
- terminology • 47
 - access types • 61
 - comparison • 62
- test
 - DCOM configuration • 394

text

- replacing • 136
- searching for • 136

Tool Editor • 111

toolbars • 119

- add to windows • 171
- adding tools • 173
- attach to windows • 173
- conventions • 175
- create • 171
- customize • 122
- editing • 111
- logic • 185
- properties • 173

tools

- add to toolbars • 173

U

UML models • 447

Unified Modeling Language (UML) • 52, 447

Uninstall • 25

Uninstall on Microsoft Windows • 30

use

- commit modes • 244
- inheritance to reuse queries • 223
- interface layers • 444

utilities

- change management • 130

V

visual editors • 22

W

Watched Attribute pane • 420

watchpoints • 432

Web services • 399

- deployment • 399
- programming • 399
- standards • 400

WebLogic deployment • 405

WebSphere deployment • 406

WhenFetched() method • 241

WhenUpdated() method • 246

Window Editor • 111

windows

- add controls • 151
- add graphics • 175
- add menus • 163
- add toolbars • 171

applications

using in • 146

attach menu titles • 166

attach toolbars • 173

editing • 149

logic • 179

manipulate in IDE • 91

MDI • 148

Output Window • 93

Project Workspace • 96

properties • 150

SDI • 148

windows, • 148

writing

SQL • 231

X

XML Metadata Interchange (XMI) • 447