AppBuilder

By Magic Software Enterprises

Magic Software AppBuilder

Version 3.2

Debugging Applications Guide

Corporate Headquarters:

Magic Software Enterprises
5 Haplada Street,
Or Yehuda 60218, Israel
Tel +972 3 5389213
Fax +972 3 5389333

© 1992-2013 AppBuilder Solutions
All rights reserved.
Printed in the United States of America.
AppBuilder is a trademark of AppBuilder Solutions. All other product and company names mentioned herein are for identification purposes only and are the property of, and may be trademarks of, their respective owners.

Portions of this product may be covered by U.S. Patent Numbers 5,295,222 and 5,495,610 and various other non-U.S. patents.
The software supplied with this document is the property of AppBuilder Solutions and is furnished under a license agreement. Neither the software nor this document may be copied or transferred by any means, electronic or mechanical, except as provided in the licensing agreement.
AppBuilder Solutions has made every effort to ensure that the information contained in this document is accurate; however, there are no representations or warranties regarding this information, including warranties of merchantability or fitness for a particular purpose. AppBuilder Solutions assumes no responsibility for errors or omissions that may occur in this document. The information in this document is subject to change without prior notice and does not represent a commitment by AppBuilder Solutions or its representatives.

1. Debugging Applications Guide . . . 2
  1.1 Introduction to Debugging . . . 2
    1.1.1 Available Diagnostic Flags . . . 2
    1.1.2 View Watch . . . 3
    1.1.3 RuleView . . . 4
  1.2 Debugging Java Applications . . . 5
    1.2.1 AppBuilder Java Tracing Overview . . . 5
    1.2.2 Installing RuleView for Java . . . 8
    1.2.3 Getting Ready to Debug in Java . . . 9
    1.2.4 Starting RuleView . . . 11
    1.2.5 Understanding the RuleView for Java Interface . . . 12
    1.2.6 Running RuleView . . . 24
    1.2.7 Troubleshooting . . . 35
  1.3 Debugging .NET applications . . . 36
    1.3.1 Tracing Options . . . 37
    1.3.2 Getting Ready to Debug in .NET . . . 37
  1.4 Debugging C Applications with RuleView . . . 38
    1.4.1 Getting Ready to Debug in C . . . 38
    1.4.2 Starting RuleView for C . . . 39
    1.4.3 Understanding the RuleView Interface . . . 40
    1.4.4 Debugging with RuleView for C . . . 44
    1.4.5 Exiting RuleView for C . . . 49
  1.5 Debugging Batch Applications . . . 50
    1.5.1 Batch RuleView Commands and Function Keys . . . 50
    1.5.2 Batch RuleView Panels . . . 51
  1.6 Debugging CICS and IMS Applications . . . 55
    1.6.1 Mainframe RuleView Panels . . . 55
    1.6.2 Mainframe RuleView Commands and Function Keys . . . 55
    1.6.3 Rule Selection List HPR0 . . . 56
    1.6.4 Mainframe RuleView Breakpoint Selection . . . 57
    1.6.5 Mainframe RuleView Source Code Display . . . 58
    1.6.6 Mainframe RuleView View Display . . . 58
    1.6.7 Mainframe RuleView Test Data Maintenance . . . 59
    1.6.8 Mainframe RuleView Test Data Input . . . 60

Debugging Applications Guide

This guide explains how you can use AppBuilder tools to debug your applications.

Debugging is the process of finding and fixing execution time errors (when the application attempts to perform a prohibited task) and design logic errors (with correct code but not what was intended). AppBuilder provides the following tools for debugging applications:

- View Watch
- RuleView

This guide is intended for consultants and customers who understand distributed environments and who are familiar with the concepts of distributed applications but may not know how to implement these types of applications with the AppBuilder product. Some experience with debugging applications in Windows is assumed.

The following subjects are covered:

- Introduction to Debugging
- Debugging Java Applications
- Debugging .NET applications
- Debugging C Applications with RuleView
- Debugging Batch Applications
- Debugging CICS and IMS Applications

Introduction to Debugging

Introduction to Debugging

Debugging is the process of finding and fixing execution time errors (when the application attempts to perform a prohibited task) and design logic errors (with correct code but not what was intended). AppBuilder provides the following tools for debugging applications:

- View Watch
- RuleView

Both View Watch and RuleView are available for Java applications. A slightly different version of RuleView is available for C applications.
This guide is intended for consultants and customers who understand distributed environments and who are familiar with the concepts of distributed applications but may not know how to implement these types of applications with the AppBuilder product. Some experience with debugging applications in Windows is assumed.

Available Diagnostic Flags

Available Diagnostic Flags

ASSERT VIEW IDENTITY

ASSERT_VIEW_IDENTITY can be added to the hps.ini file, and the possible values for this setting are YES and NO. When set to YES, every view class has a getHash method generated and an assertIdentity method is invoked to verify the rule input and output view. This verification prevents a rule being executed with a view class being different from the view class the rule was compiled with.
When set to NO, method getHash is not generated in a view class and assertIdentity method is not invoked.

GENERATE RULE CALLS TRACE

This is an Hps.ini file setting that generates DEBUG level logging statements to log rule calls. To enable this functionality, set the value of GENERATE_RULE_CALLS_TRACE to YES in the [CodegenParameters] section of the Hps.ini file. This setting can have the values YES or NO. When set to YES, debug level logging statements are generated to log the beginning and end of the rule execution. When set to NO, the additional logging statements are not generated. At runtime the logging information will be placed in the log file if APP_LOGGER level is set to DEBUG for log4j or FINE for Java logging.

ALWAYS GENERATE USER TRACE

This setting is used in the Hps.ini file to generate user TRACE statements when debug info generation is disabled.
To enable this functionality, change the setting ALWAYS_GENERATE_USER_TRACE in the [CodegenParameters] section of the Hps.ini file. This setting can have the values YES or NO. When set to YES, user TRACE statements from the rule code are generated regardless of debug info generation setting. When set to NO, user TRACE statements are generated if debug generation is enabled. They are not generated if debug

generation is disabled. The command-line flag GENTRACE can be specified for codegen. Setting this flag is equivalent to setting ALWAYS_GENERATE_USER_TRACE to YES.

GENERATE IO VIEW TRACE

This setting is used in the HPS.INI file to generate input/output views debug TRACE statements.
To enable this functionality, change the setting GENERATE_IO_VIEW_TRACE in the [CodegenParameters] section of the HPS.INI file. This setting can have the values YES or NO. When set to YES, the input view trace is generated in the beginning of the rule and the output view trace is generated at the end of the rule.

View Watch

View Watch Start Settings

To make View Watch available, complete the following steps:

1. Click Windows - Start menu and select ***All Programs > AppBuilder > Configuration > Management Console***.
2. In the AppBuilder Configuration dialog, right-click *Java* under Clients and select *Edit Appbuilder.INI* from the pop-up menu.
3. When the appbuilder.ini file loads, select the NC tab.
4. Set the Show View Watch Window setting to True to access this debugging tool.

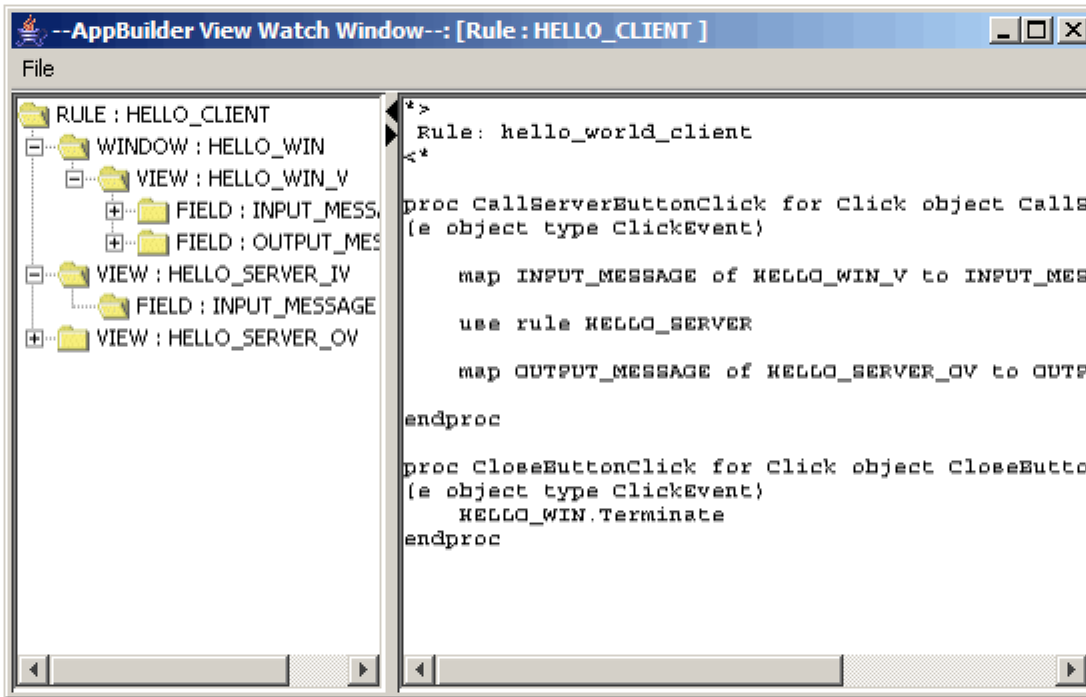
Setting View Watch in AppBuilder.ini

Name	Value
CLOSE_DETACHED_RULES_W...	TRUE
NEW_JVM_FROM_MENU	TRUE
FRAME_TYPE	WINDOW
QUERY_AUTHENTICATION_ON...	FALSE
AUTH_TYPE	EXIT
PROPAGATE_NULL_TO_DATA...	FALSE
SHOW_VIEW_WATCH_WINDOW	TRUE
EXCEPTION_ON_UNSUPPORT...	TRUE
ENABLE_CLEAR_WINDOW_CH...	FALSE
SHOW_ZERO_ON_NULL	FALSE
CHAR_COORDINATE_FONT	CCS_FONT
IMAGES_DIRECTORY_URL	http://someserver.acompany.com/images/

Debugging applications with View Watch

You can use View Watch to display the views in a Java application in a tree structure. The structure of the application is displayed in the left pane, with the rule source in the right pane. The contents of a view can be displayed in the left pane of the View Watch window as the application runs. The values in a view cannot be modified.

View Watch Window



To access View Watch when a Java application is running, press Ctrl + F9.

RuleView

RuleView

RuleView enables you to monitor and debug one statement at a time when you run an application. With a few optional settings, the debugger can start automatically when you run an application. This guide explains how to use the RuleView debuggers for debugging an application.

RuleView Start Settings

To use the RuleView debugger, you must set a value in the initialization (INI) file for the Construction Workbench. You can do this in the Management Console, or you can manually adjust settings in the ini file.
For Java applications, change the setting in the following ini file, which is located in the JAVA/RT directory of the AppBuilder installation:

- ini file name: appbuilder.ini
- section name: TEST
- setting: DEBUG_START

For C applications, change the setting in the following file, which is located in the directory where AppBuilder is installed:

- ini file name: hps.ini
- section name: AE Runtime
- setting: HPSRT_DEBUG_START

The valid values are ***QUERY*** , ***TRUE*** , or ***FALSE*** ., which are the same for both settings.

- ***QUERY*** - This is the default setting. The system prompts you select whether or not you want to start RuleView when the application is run.
- ***TRUE*** - The system always starts RuleView when the application is run without asking.
- ***FALSE*** - The system does not start RuleView when the application is run. This setting is always set on production client workstations.

The RuleView start setting stays in effect regardless of how you start the application?from the Construction Workbench or outside the Construction Workbench.

Debugging applications with RuleView

You can examine the contents of views attached to rules, set breakpoints, skippoints (C only), and watchpoints (depending on whether the application is running in C or Java) and look at the value of variables as the code is executed, one statement at a time. There are two versions of the RuleView debugger, one for C and one for Java, with only slight differences between them. Each version of the RuleView debugger is discussed separately:

- Debugging Java Applications
- Debugging C Applications with RuleView

Debugging Distributed Applications

The Java version of RuleView supports distributed applications debugging. No additional modules or configuration is required to enable this feature. The main rule should be started on the client as usual, and AppBuilder runtime support and RuleView take care of any remote calls. Such calls look as if they are simple local subrule calls. as an integrated product. This is not correct. SourceFire is not integrated with Protection Center 2.1.

Debugging Java Applications

Debugging Java Applications

You can now use Java tracing in conjunction with RuleView for Java to debug Java applications locally. This section discusses the tools used for tracing, logging, and debugging Java applications. You can use RuleView for Java to look at rules deployed as Java applications and see the available functions in the RuleView interface for Java application development. Before debugging the application, verify the syntax and logic of the individual rules and resolve any preparation problems. The process of verification is explained in the *Developing Applications Guide* and the troubleshooting section covers preparation problems in the *Deploying Applications Guide*. For information about debugging C applications, refer to Debugging C Applications with RuleView.
This chapter covers the following Java topics:

- AppBuilder Java Tracing Overview
- Installing RuleView for Java
- Getting Ready to Debug
- Starting RuleView
- Understanding the RuleView for Java Interface
- Running RuleView
- Troubleshooting

AppBuilder Java Tracing Overview

AppBuilder Java Tracing Overview

In addition to a number of different features and enhancements, AppBuilder includes a robust and extensible logging and tracing framework for Java applications. This new feature allows the AppBuilder clients of Java applications to choose from a number of third-party tracing implementation API's.
Java applications built with AppBuilder (procedural) rules use one framework to generate the set of classes and JAR file for the application. These applications can use a third-party logging API for tracing. Java applications built with the newer AppBuilder OO classes use a newer Java framework to generate a different set of classes and JAR file. These applications can use either a third-party logging API or AppBuilder's logging API for Java applications.

Using a Third-Party Logging Framework

A logging avenue is available to both those building Java applications from (procedural) rules and those building Java applications from AppBuilder OO classes. Both Procedural and OO AppBuilder frameworks generate calls using a delegate mechanism through the Apache commons-logging framework. This third-party framework includes a generic (abstract) set of interfaces that are wrappers around the most commonly-used industry-standard Logging Framework available. These include:

1. Log4J
2. Java 1.5 Logging

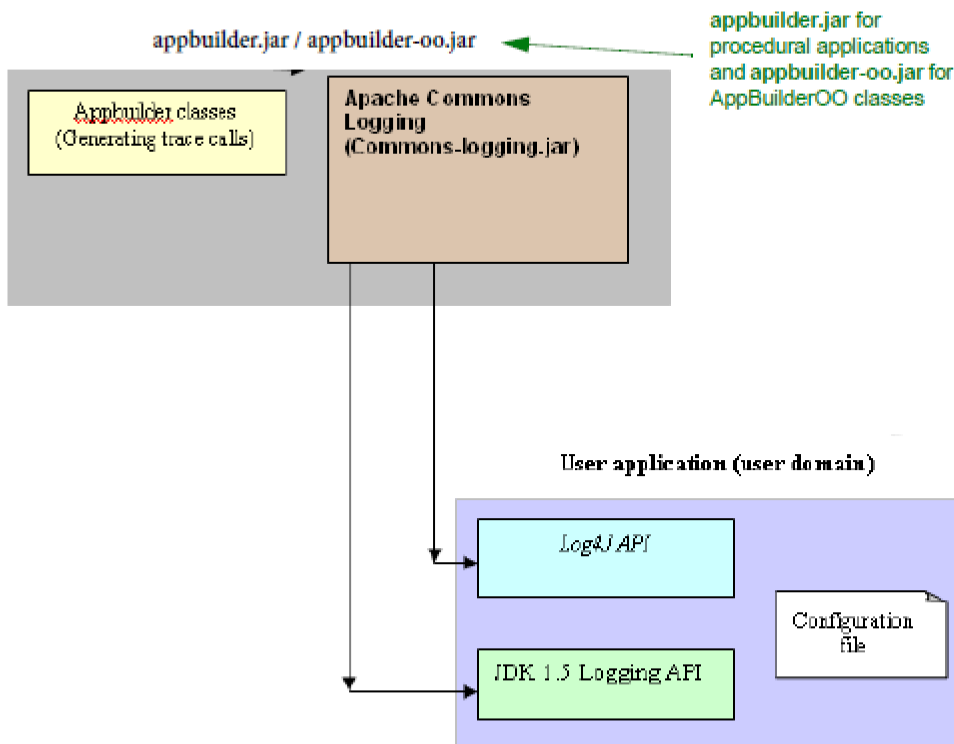
Additional File Dependencies

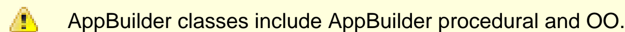
The appbuilder.jar file requires the commons-logging.jar on user environment for compiling (building) and running Java applications. This third-party jar file is shipped with the AppBuilder install and is added in the user's class path environment variable. This jar file can be located at the following AppBuilder install directory [Appbuilder\java\rt\commons-logging.jar].
The new tracing framework additionally requires that the user environment provide a concrete logging API in their environment and their properties file, as shown in Procedural and OO AppBuilder frameworks.
AppBuilder installs log4j.jar on the user environment and adds it to the user's class path environment variable. This file can be located at the following AppBuilder install directory: [Appbuilder\java\rt\lib\log4j.jar].
The following shows the two main concrete logging APIs and their properties files.
AppBuilder installs log4j.jar with its setup and uses it as the default logging API.

Logging APIs

Concrete Logging API	**API Library**	**Configuration (Properties file)**
Log4J	log4j-[version].jar (to be added in class path)	log4j.properties (default sample file shipped with appbuilder [Appbuilder\java\rt\log4j.properties]) This file should be present in the application root folder for Log4J to read.
JDK 1.5 Logging	JDK 1.5	logging.properties (default sample file shipped with appbuilder [Appbuilder\java\rt\logging.properties]) This file should be present in [JDK\jre\lib\logging.properties] folder on the user environment for JDK Logging to function properly.

Procedural and OO AppBuilder frameworks





AppBuilder classes include AppBuilder procedural and OO.

Important points regarding Java Tracing

The new Java Tracing framework in AppBuilder requires the following jar file in the environment class path:

commons-logging.jar (http://jakarta.apache.org/commons/logging/)

The commons-logging.jar file is provided with AppBuilder and is installed in the following location:
C:\AppBuilder\JAVA\RT\lib\commons-logging.jar.
In addition, you must add additional jar files (logging APIs) to their environment depending on which third party API you want to use. For example, to implement the AppBuilder logging with Log4J you would have to add the log4j.jar file in the class path. AppBuilder installs both the commons-logging.jar and log4j.jar with its setup.
The new Java Tracing framework does not use the [Tracing] settings in appbuilder.ini or appbuildercom.ini. Adding those sections in "ini" files or modifying them does not affect Java application tracing.
The new tracing framework consists of two main Loggers for AppBuilder users.

APP_LOGGER	For generating user application level logging
COMM_LOGGER	For communications client traces and logging

These loggers and their respective output handler configurations are defined in the properties files that are shipped with appbuilder.jar (log4j.properties and logging.properties); therefore it is important that these files are not removed and that the existing content not be deleted. However, users can change settings such as levels of logging, output format, and output locations.
The new Tracing framework allows AppBuilder users to choose from a number of logging APIs. Some of the third-parties' logging APIs that can be used with AppBuilder are as follows:

- Log4J Logging API
- JDK 1.5 Logging

For a complete list, please see http://jakarta.apache.org/commons/logging/guide.html#Introduction.

When applications are deployed on a third-party application server, the commons-logging.jar needs to be available on the application server environment. For the user environment based logging API (e.g. log4j or jdk14), that third party library and its configuration should be present along with user application.

AppBuilder Java Logging Settings and Configurations

AppBuilder Java Logging framework collects settings and configurations from the properties file that are provided on the user environment.

Java Logging Configurations

Log4j API	If the user environment is configured for using Log4j as the concrete logging API, then the application will load settings from the log4j.properties file. For a complete list of different properties and settings, please see the following url: http://logging.apache.org/log4j/docs/manual.html
JDK 1.5 Logging	If the user environment is configured for using JDK1.5 logging as the concrete logging API, then the application will load settings from the logging.properties file. For a complete list of the properties and settings, please see the following url: http://java.sun.com/javase/reference/index.jsp.

Following are the main set of properties that are used:

Properties used for logging configurations

Category	Attributes	Attribute Detail	Log 4J Logging (log4j.properties)	JDK 1.5 Logging (logging.properties)
Loggers (Trace Handlers)	APP_LOGGER	For generating user application level logging	log4j.logger.APP_LOGGER	APP_LOGGER
Tracing Levels	OFF	No Tracing	log4j.logger.APP_LOGGER=OFF	APP_LOGGER.level = OF
	ALL	All Tracing enabled	log4j.logger.APP_LOGGER=ALL	APP_LOGGER.level = AL
	INFO	User-specified trace calls from rules	log4j.logger.APP_LOGGER=INFO	APP_LOGGER.level = IN
	DEBUG	User traces and traces from rule calls and events	log4j.logger.APP_LOGGER=DEBUG	APP_LOGGER.level = FI
	TRACE	User traces when user calls the trace() statement in OO Language	log4j.logger.APP_LOGGER=TRACE	APP_LOGGER.level = FI
	WARN	Warning messages	log4j.logger.APP_LOGGER=WARN	APP_LOGGER.level = W

	ERROR	Error messages	log4j.logger.APP_LOGGER=ERROR	APP_LOGGER.level = SE
Tracing Output Locations	File	For generating trace output to a File	log4j.appender.APP_LOGGER=org.apache.log4j.RollingFileAppender	APP_LOGGER.Handler = java.util.logging.FileHandl
	Console	For generating trace output to Console	log4j.appender.APP_LOGGER=org.apache.log4j.ConsoleAppender	APP_LOGGER.Handler = java.util.logging.ConsoleH
Tracing Output Formats	Simple Text	For generating trace output in simple text format	log4j.appender.APP_LOGGER.layout=org.apache.log4j.SimpleLayout	java.util.logging.ConsoleH = java.util.logging.SimpleF
	XML	For generating trace output in xml output	log4j.appender.APP_LOGGER.layout=org.apache.log4j.XMLLayout	java.util.logging.ConsoleH = java.util.logging.XMLFo

Steps for implementing Log4J as the underlying logging API

Complete the following steps:

1. If the commons-logging.properties file is present in the class path, it does not contain a parameter like org.apache.commons.logging.Log=org.apache.commons.logging.impl (that is, some API other than Log4J).
2. Make sure that in the classpath or (in the loading process for the application class loader) Log4J.jar is present.
3. Make sure that log4j.properties is present in the classpath or is loaded with the application.

For a complete list of settings for commons-logging.properties, please see the reference, http://jakarta.apache.org/commons/logging/guide.html.

Steps for implementing JDK 1.5 Logging as the underlying logging API

Complete the following steps:

1. The user application is running under JDK1.5 or later versions of runtime.
2. Make sure that if there exists a commons-logging.properties file in the class path, it does not contain a parameter such as this: org.apache.commons.logging.Log=org.apache.commons.logging.impl (that is, some API other than JDK14 logging).
3. Make sure that in the classpath or (in the loading process for the application class loader), there is no Log4J.jar.

For a complete list of settings please see the reference, http://jakarta.apache.org/commons/logging/guide.html.

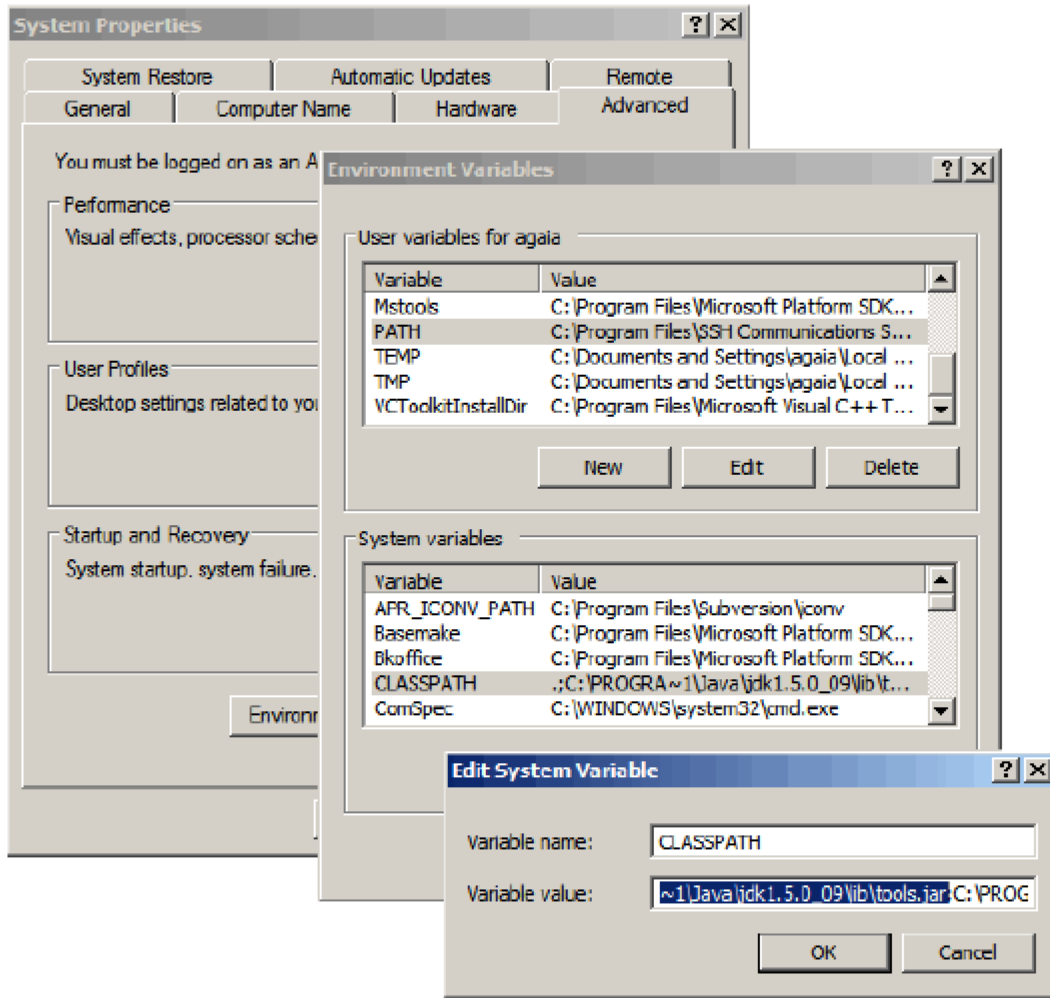
Installing RuleView for Java

Installing RuleView for Java

The RuleView for Java source debugger is built on top of the Java Platform Debugger Architecture (JPDA) from Sun Microsystems. The JPDA is included with the Java JDK version 1.5. AppBuilder's RuleView is based off of Java JDK version 1.5. The only thing that is necessary is to add the Java tools.jar library that is distributed with the JDK to your classpath.
The JPDA must be installed on the development workstation. The tools.jar library is located inside of the base directory of the JDK (where you installed Java Version 1.5) in the lib subdirectory. To get RuleView to work properly, follow the steps:

1. Add the absolute path of this jar file to the User's or the Global System Classpath.Open ***My Computer > Properties > Advanced > Environment Variables > CLASSPATH*** from System Variables and click ***Edit*** .
2. In the Variable value field, add the path to the tools.jar file: *C:\PROGRA~1\Java\jdk1.5.0_09\lib\tools.jar.*
3. Click ***OK*** .

Update Java Classpath in the Environmental Variables



In the Environmental Variables window, press ***OK*** , and then ***Apply*** . The system is updated automatically.

Sometimes the system variables are not updated immediately after a change. In this case, reboot your system.

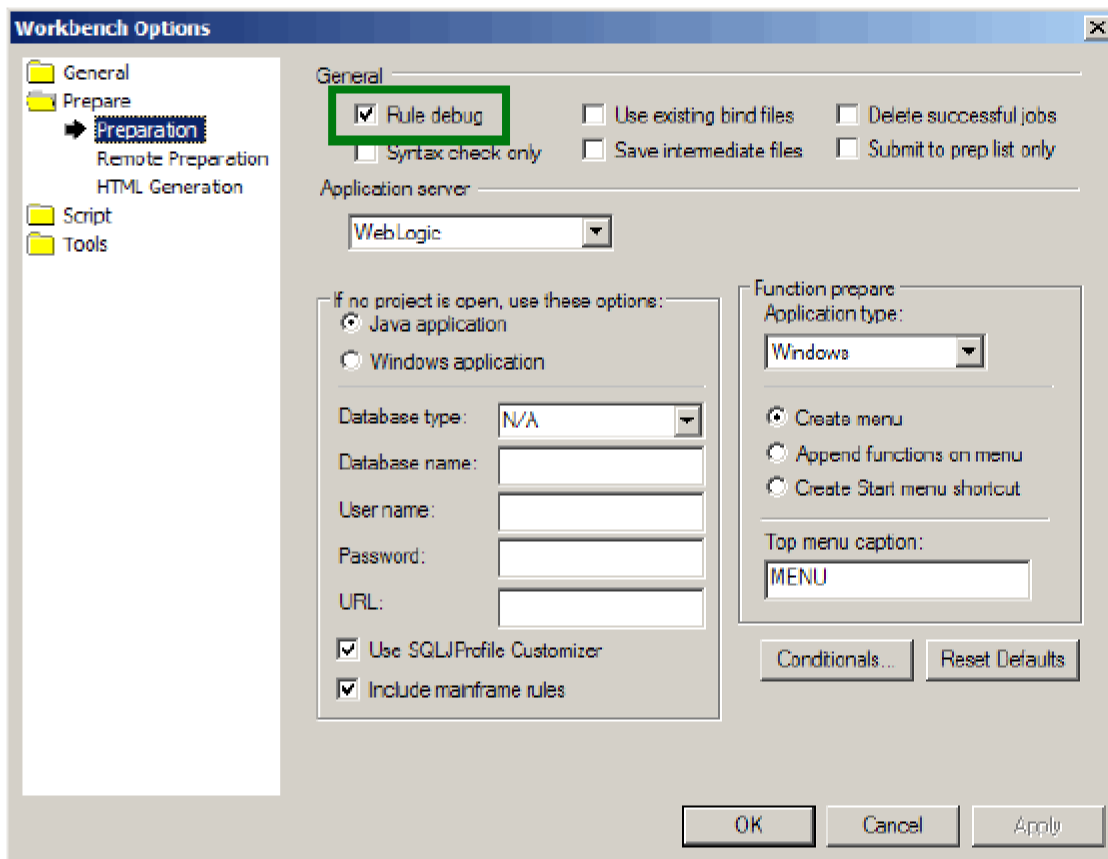
Getting Ready to Debug in Java

Getting Ready to Debug

Before running the debugger for an application, follow these steps to set the Rule Debug option.
To enable the debug option:

1. From Construction Workbench, select ***Tools > Workbench Options*** .
2. Click the ***Preparation*** section.
3. Make sure that the Rule Debug box is checked.

Preparation section of the Workbench Options dialog



Refer to the *Deploying Applications Guide* for more information.

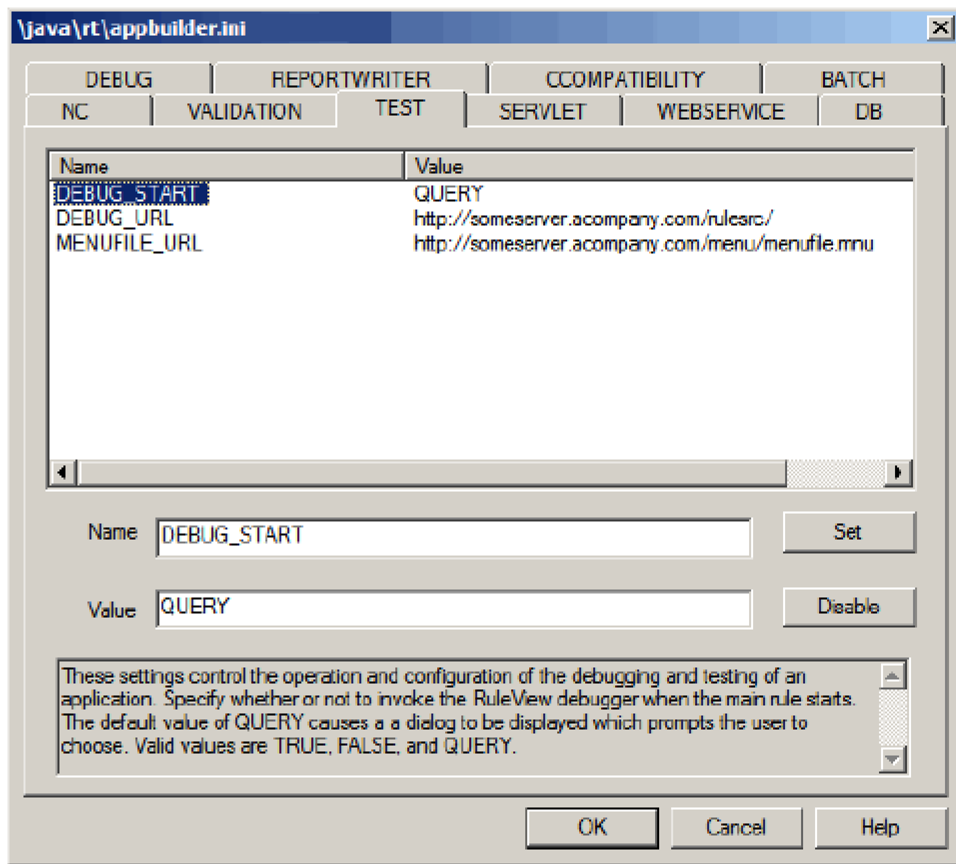
To use all the features of the RuleView debugger, make sure that all the rules of the application are prepared with the Rule Debug option. This option tells the code generator to include debug information that is necessary to perform debugging commands, such as setting a breakpoint or viewing data. If one or several rules in the application have been prepared without the Rule Debug option, see *Debugging Rules without Debug Information*.

You can also change the ini setting to enable the use of RuleView for debugging Java applications. You can manually adjust the setting in the ini file, or use the Management Console. The ini file is located in the JAVA/RT directory of the AppBuilder installation.

To enable the RuleView for Java applications:

1. Click Windows? Start menu and select ***All Programs > AppBuilder > Configuration > Management Console***.
2. In the AppBuilder Configuration dialog, right-click ***Java*** under Clients and select ***Edit Appbuilder.INI*** from the pop-up menu.
3. When the appbuilder.ini file loads, select the TEST tab.
4. Find the DEBUG_START setting. The valid values are as follows:
   - ***QUERY*** – This is the default setting. The system prompts you select whether or not you want to start RuleView when the application is run.
   - ***TRUE*** – The system always starts RuleView when the application is run without asking.
   - ***FALSE*** – The system does not start RuleView when the application is run. This setting is always set on production client workstations.

Appbuilder.ini file [TEST] section DEBUG_START setting



The debut option setting and the RuleView start setting stays in effect regardless of how you start the application---from the Construction Workbench or outside the Construction Workbench.

Starting RuleView

Starting RuleView

To debug an application, in the Construction Workbench menu, complete the following steps:

1. Select ***Run > Java*** .The Java Execution Client displays.
2. Select the function that you want to run. The system displays a dialog with the following message:*Do you wish to start the RuleView?*
3. Click ***Yes*** .

OR
To invoke it from the command line, set the DEBUG_START option of the appbuilder.ini file. Refer to Initialization Settings for more information.
To debug an individual rule developed in Java, type the following on a command line or in a command prompt window:

java appbuilder.tools.RuleView [-options] My_Rule

or

java My_Rule

where *-options* are any of the available RuleView options and *My_Rule* is the rule class name (without .class extension) to be debugged.
If you use the latter method, a dialog box with the message " *Do you wish to start the RuleView?* " appears in the standard way, depending upon the DEBUG_START option in the appbuilder.ini file.
The available options are:

- *-c* This tells RuleView to use the class name for *My_Rule* . It uses the rule long name by default.
- *-s <servlet_URL>* This tells RuleView the servlet to debug.
- *-n <ini_URL>* This tells RuleView the INI file to use to initialize the system.

The RuleView for Java does not need a Function to be prepared for you to view the correct process hierarchy. If you have changed the structure of a process, you only need to re-prepare the rules of the child entities that have been reorganized. However, if you are starting RuleView through the Execution Client, you must prepare the Function anyway.

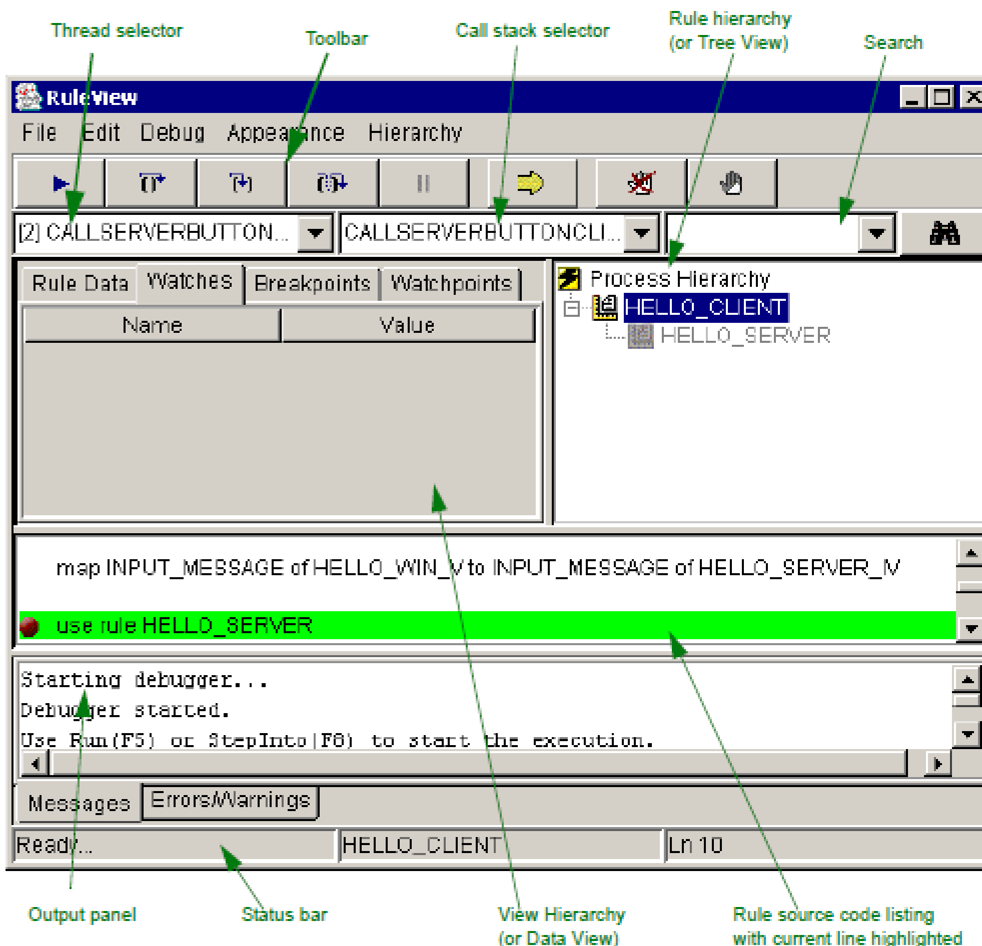
Understanding the RuleView for Java Interface

Understanding the RuleView for Java Interface

RuleView is a Rules Language source code debugger that has its own graphical user interface. When you become familiar with the user interface you can perform the debugging operations. The interface of RuleView for Java is slightly different from that of RuleView for C. This section discusses the following parts of the RuleView for Java interface:

- Terminology
- Pull-down Menu
- Toolbar
- Rule Hierarchy
- View Hierarchy
- Rule Source Code Listing
- Output Panel
- Status Bar
- Look and Feel
- RuleView Actions

RuleView for Java window



Terminology

To understand the RuleView interface, you must understand two important terms: Call Stack and Thread.

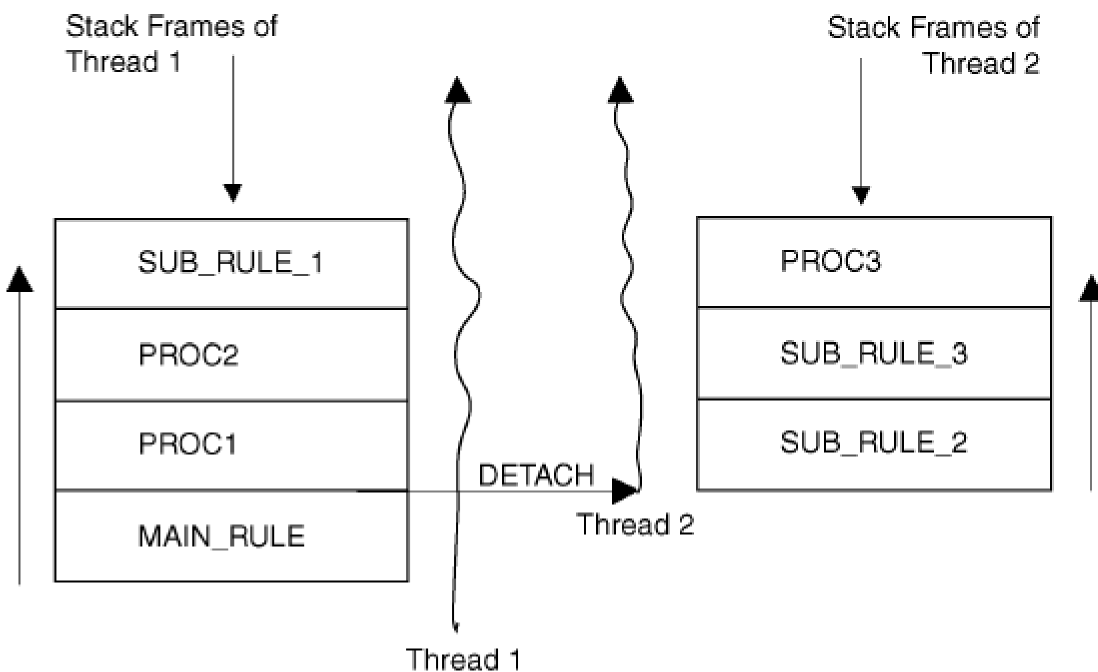
Call Stack

When an application runs, it executes local procedures and rules, beginning with its root rule. If the root rule uses another rule, the second rule is added to the list of rules and procedures that are currently executing, which is called the call stack. The second rule can use another rule or call a local procedure, which is added as well to the stack. The rule or procedure at the top of the stack is called the executing rule (procedure). When the executing rule (procedure) finishes, it is removed from the stack. Each item of the call stack list is called a stack frame. The currently executing rule or procedure corresponds to the top stack frame. The first stack frame in the list, called the bottom stack frame, refers to the main rule, the rule that began the execution. Each stack frame has a level, the number in the sequence. You do not have to deal with the stack frame levels; that is the duty of the debugger.
The data of the local procedures are stored in the stack frame. This protects them from being changed or damaged by other local procedures or from the rule. Thus, each local procedure on the stack has its own instances of its variables. Except for its own local variables, all global variables (which you have created in the rule hierarchy or declared in the rule to which this procedure belongs) can be accessed from the local procedure. All these variables are called *visible variables* . RuleView displays only visible variables at any call stack level.
Also, the stack frame holds information about the *current location* in rule source, or rule line that is to be executed next. Each stack frame has its own current location.
In order to distinguish stack frames easily, they are given *names* . The name of the stack frame coincides with the name of the corresponding rule or procedure. The stack name is the same as the name of the rule or procedure to which this stack corresponds. For example, the stack of a rule named MAIN_RULE has a name "MAIN_RULE" and a stack of a local procedure PROC1 is named "PROC1". If there are several procedures with the same name or some procedure is called recursively, there are several stack frames on the stack with the same name. They are distinguished by their levels, which is seen by the placement of the stack frames in the stack combo box. The top stack frame (the stack frame of the executing rule or procedure) is the top element of the combo box.

Thread

RuleView for Java supports threads for multi-threaded applications. A thread is the smallest unit of parallel program execution. It is like a process for C applications. If the Java application contains a USE RULE DETACH statement, the called non-window rule is executing in a new thread parallel to the caller. For window rules the behavior is more complicated due their event-driven nature. There are numerous system threads in the runtime support environment, such as an event-dispatcher thread for GUI applications. All handlers from window events are executed in this thread. Each thread has its own call stack, or a list of rules and procedures currently executing in this thread.
RuleView displays only those threads that run through the rule code; if a thread is running but not in the rule, RuleView does not display it.
RuleView does not allow suspension of one or more threads of the application without suspending the entire application. Thus, you can use the debugging commands to suspend the threads (at appropriate locations) and to inspect the stacks and the data for the threads.
The *thread name* is the name of the bottom stack of this thread; thus, it usually equals the main rule name. However, there could be several threads with the same name. To distinguish them, a unique thread identifier is introduced. It is shown in square brackets to the left of the thread name.
The stack name is equal to the name of the rule or procedure to which this stack corresponds. For example, the stack of a rule named MAIN_RULE has a name "MAIN_RULE" and a stack of a local procedure PROC1 is named "PROC1".

Stack frames of threads



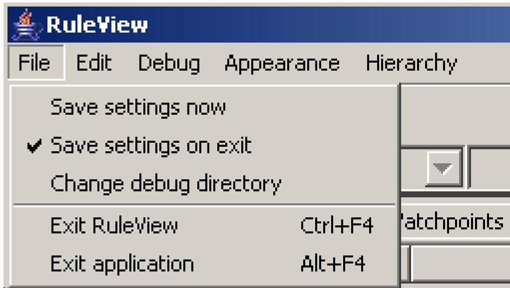
Pull-down Menu

Pull-down menus provide controls for debugging actions. Many functions are also available in pop-up menus and from buttons in the Toolbar. Use the RuleView toolbar to quickly access the RuleView Actions.

File menu

Use the File menu to exit the application or RuleView and manage the user interface settings. Individual menu items are described in Description of File menu - RuleView for Java.

File menu - RuleView for Java



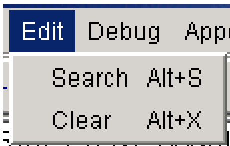
Description of File menu - RuleView for Java

Menu Item	Description
Save settings now	To save all the RuleView settings (like the window size and position) immediately. Refer to Saving and Restoring Settings.
Save settings on exit	If checked, RuleView saves its settings automatically when it exits. Refer to Saving and Restoring Settings.
Change debug directory	To save the debug settings to an externally-available file, you can use this command to specify the location of that debug directory.
Exit RuleView	To just exit RuleView without affecting the execution of the application. Refer to Exiting RuleView Only.
Exit application	To exit the application being debugged. Refer to Exiting the Application.

Edit menu

Use the Edit menu to search rule source code for an alphanumeric string. Individual menu items are described in Description of Edit menu - RuleView for Java.

Edit menu - RuleView for Java



Description of Edit menu - RuleView for Java

Menu Item	Description
Search	To perform a string search.
Clear	To clear the search string.

Debug menu

The Debug menu has most of the debugging actions, from stepping through the execution to setting or clearing breakpoints and watchpoints. Individual menu items are described in Description of Debug menu - RuleView for Java.

Debug menu - RuleView for Java

Description of Debug menu - RuleView for Java

Menu Item	Description
Go	To resume the previously suspended application. For more information, see *Resuming* .
Step over	To step to the next valid (executable) rule line without stopping in any subrules or procedures. For more information, see *Stepping Over* .
Step into	To step to the next valid (executable) rule line at any call stack level. For more information, see *Stepping Into* .
Step out	To step out of the current subrule or procedure. For more information, see *Stepping Out* .
Pause	To stop the execution of a rule without setting a breakpoint. For more information, see *Pausing Execution* .
Show current location	To find the current location of the executed code. For more information, see *Showing the Current Location* .
Set breakpoint	To set a breakpoint at the currently highlighted location in the source code window or at the specified line. For more information, see *Setting a Breakpoint* .
Enable/Disable breakpoint	To disable or enable the breakpoint that is set to a line in a rule. The breakpoint that is being disabled is ignored by the debugger. For more information, see *Enable or Disable a Breakpoint* .
Clear breakpoint	To remove a breakpoint. For more information, see *Clearing a Breakpoint* .
Set watchpoint	To set a watchpoint on a variable. For more information, see *Setting a Watchpoint* .
Clear watchpoint	To remove a watchpoint. For more information, see *Clearing a Watchpoint* .
Exceptions	To handle exceptions. For more information, see *Handling Exceptions* .

Appearance menu

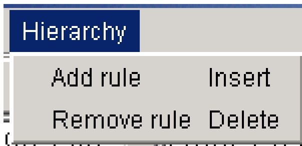
Use the Appearance menu to control the Look and Feel of the user interface of the debugger. They do not affect the performance of the debugger. The default is Windows.

Appearance menu - RuleView for Java

Hierarchy menu

Use the Hierarchy menu to add a rule or to remove a rule from the rule hierarchy. This is useful if some rule is prepared without the debug information and you want to set a breakpoint, for example, to one of its subrules.

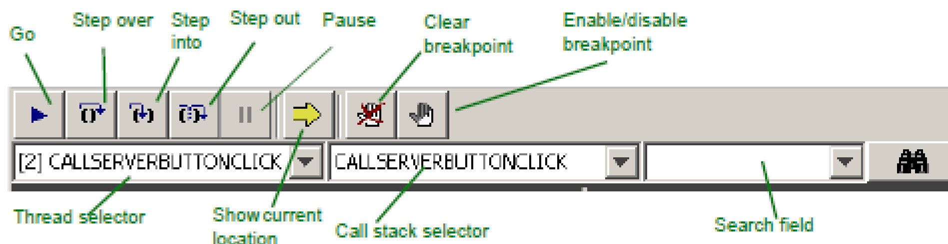
Hierarchy menu - RuleView for Java



Toolbar

Frequently used functions can be run by clicking the buttons in thtoolbarolbar. These same functions are available in pop-up menus and from the Pull-down Menu. Use the RuleView toolbar to quickly access the RuleView Actions.

RuleView for Java toolbar



Thread Selector

This combo box contains all the threads that are executing in the application, as described in Thread. Threads are not ordered. However, when RuleView gains control after finishing a debugging operation, the thread currently selected (current thread) is one where last event occurred (breakpoint hit, step finishes, Watchpoint triggered, etc.).
You can choose any thread from the combo box to view its call stack. When the current thread is changed, the call stack combo box is automatically updated to reflect the change and displays the selected thread stack frames. Similarly, the Rule Hierarchy selects the rule corresponding to the currently selected call stack frame in the current thread.

Call Stack Selector

This combo box contains the call stack of the current thread. When RuleView updates this call stack combo, the currently selected item is the topmost stack frame, or the current stack frame or stack frame of the currently executing procedure or rule.
You can switch between stack frames to see the corresponding set of visible variables and current location in rule source. The current location is highlighted green in the rule source window. When you change the current stack frame, the rule hierarchy is automatically updated to display the rule this stack frame corresponds to and vice versa. When you click some rule in the rule hierarchy, the call stack combo box is updated to display the topmost stack frame corresponding to the selected rule. It depends on the current location whether it is a frame of a rule or a procedure. If there is no corresponding stack frame, this combo box is disabled (grayed).
There is a special case to the use of stack frames. When they are associated with a rule that has no debugger information, such frames cannot access data and have no location except a procedure they represent in a stack. They are used only to display call stack structure. However, they are associated with a rule and, as with usual frames, affect rule hierarchy selection pointing to their rule. The source code list and data views are cleared for these frames. They are also highlighted with color.

Search

You can use the search engine to search for an alphanumeric string in the rule source code. The source code search engine is a text field to the right of the Rule Stack combo box. (See RuleView for Java toolbar.) Type the string to look for and press *Enter* .
If found, the first source line containing the string is selected. Press Alt+S or Edit > Search to move to the next occurence of the typed string. Press Edit > Clear Search or Alt+X to clear the search combo box.
To display the Advanced Search toolbar, press Alt+S or Edit > Search before typing the string in the search combo box. It includes forward and backward search capabilities with the option of performing case-sensitive searches. (By default, search is case-insensitive.)

Rule Hierarchy

The rule hierarchy is displayed in the upper right corner of the window. The rule hierarchy shows the names and levels of rules in the application. The rule hierarchy tree reflects rule state during execution. There are three possible states:

- *Rule is loaded.* The rule is displayed using the normal tree node styles (which strongly depend on the current look and feel). Such a rule being selected displays its source code with executable lines highlighted. If the rule is in the current call stack, the data are also displayed.
- *Rule is not loaded.* The rule is highlighted with color; it looks like disabled text. Such a rule displays only its source code with no difference between executable and other lines.
- *Loaded rule compiled with no debug information.* The rule is displayed as in the case of the rule not loaded, but using italic font. Rules of this kind do not provide any information.

You can add or remove rules from the hierarchy using the Hierarchy menu items. To display the source code in the rule source window, click the rule name that you want to view in the hierarchy. Refer to RuleView for Java window for the location of the rule hierarchy and the rule source code listing.
Generated and built-at-runtime hierarchies might differ. To improve performance, RuleView constructs a rules hierarchy step by step. Subrules of a rule are added to a hierarchy only when a rule is being loaded. However, if you need some rule to appear in a hierarchy before its parent is executed, double-click the parent rule in the rules hierarchy. This forces RuleView to load all the subrules of the selected rule if debug information for it is available. If it is not available, you can add subrules manually or perform a trace into the commands until the subrule you need is executed, and then it is added to the hierarchy automatically.

View Hierarchy

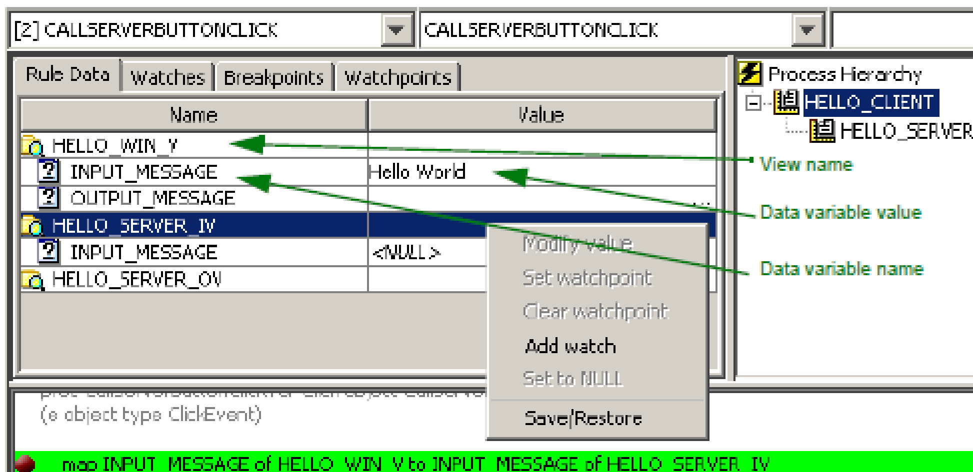
This advanced tabbed pane contains four information tabs:

- Rule Data Tab
- Watches Tab
- Breakpoints Tab
- Watchpoints Tab

If any value is too long to display in the tab, only the beginning is displayed and provided with a tooltip that shows the complete value. Refer to RuleView for Java window for the location of the view hierarchy window.
All of these tabs can be detached into a separate window. This helps view the information from multiple tabs at one glance. To detach a tab, right-double-click on its header. The new window that contains the information for that tab appears. To attach it back to the tabs simply close the window. Each tool in this area is operated through its own pop-up menu.

Rule Data Tab

This tab, displayed by default, displays all the rule data variables (views and fields) associated with the stack frame currently displayed in the Call Stack combo box. If the rule being displayed is not loaded (Call Stack combo box disabled), the view hierarchy (or data view) tab is blank. The data variable includes the name and the value as shown in Rule data tab.

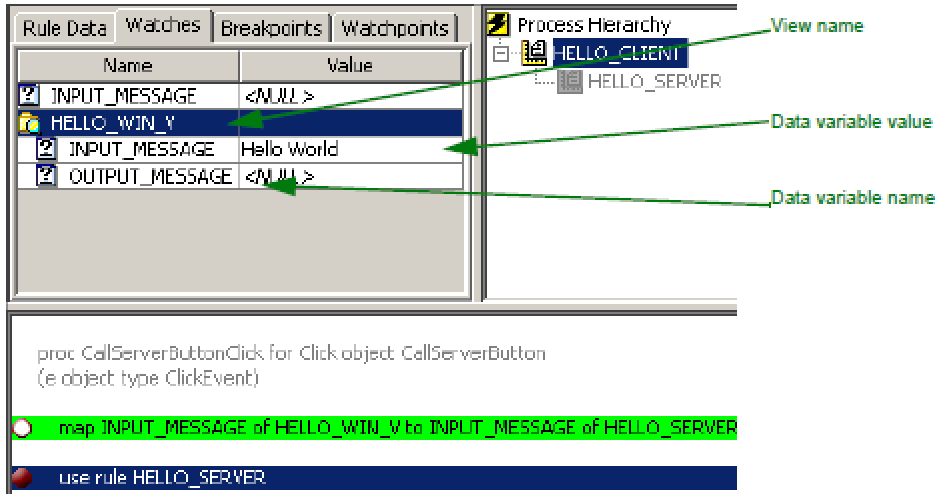


Rule data tab

If a rule handles a large amount of data, it could slow down RuleView performance, since RuleView reads rule data each time the application stops. You can use Watches Tab to specify a subset of views or fields to watch.
RuleView displays variable values using the system default format. The two settings used to format the date and time are available in the [NC] section of the appbuilder.ini file. DATE variables are formatted according to the value of the DEFAULT_DATE_FORMAT setting. TIME variables are formatted according to the DEFAULT_TIME_FORMAT setting. The date and time formatting is set for the entire RuleView window, including the Watches Tab.

Watches Tab

This tab displays variables that have been specified to watch. If the application contains a lot of data, it is inconvenient to view all the variables at one time. More frequently, you need to watch the values of several variables of interest. With watches, it is possible to add variables to the Watches tab using the pop-up menu on the Rule Data tab view or field (see Viewing and Modifying Variables for more information) and selecting *Add Watch* . The variable selected is added to the Watches tab. Displaying the full names of the variable might be very difficult, so only short names are displayed, as in the All Data tab. In addition, every variable has a tooltip displaying the complete name. Refer to Watches tab for an example.



Watches tab

The Watches tab displays names differently from the Rule Data tab because of the possibility of names clashing. For example, in the Watches tab, it is possible to have the same variable names but they would be of different types. To distinguish such variables, the complete specification of the variable name displays as a tooltip for the variable in the Watches window:

<rule name>#<stack frame name>:<variable name>

Expand non-leaf nodes to see the value of the fields of previously selected views. This is a tree display, which can be browsed by expanding or collapsing the branches of the tree.

Breakpoints Tab

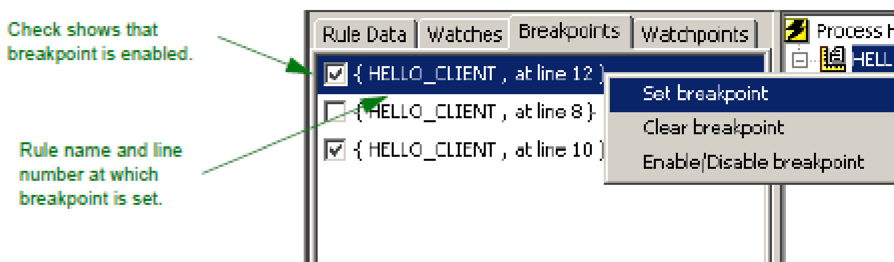
This tool displays all the breakpoints for the current debug session. Each row in the breakpoint tab consists of:

- Check box that displays and controls the status of the breakpoint. If a breakpoint is enabled, the box is checked; otherwise the box is unchecked.
- String of the following format:

{ <RULE_NAME>, at line <LINE_NUMBER> }

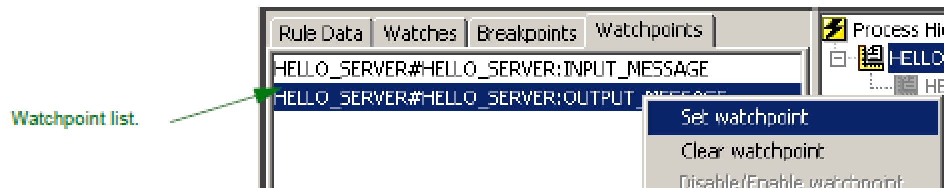
where <RULE_NAME> is the name of the rule where the breakpoint is set and <LINE_NUMBER> is the breakpoint target line.

Breakpoints tab



Watchpoints Tab

This tab displays all the watchpoints for the current debug session.



Watchpoints tab

Each row in the list consists of a string with the following format

<fldSpec>::=<name>.<fldSpec> | <name>[<index>].<fldSpec> | <name>

where <name> is a name of a view or field, and <index> is an index of viewable elements that represents a data object to which this watchpoint is targeted.

Rule Source Code Listing

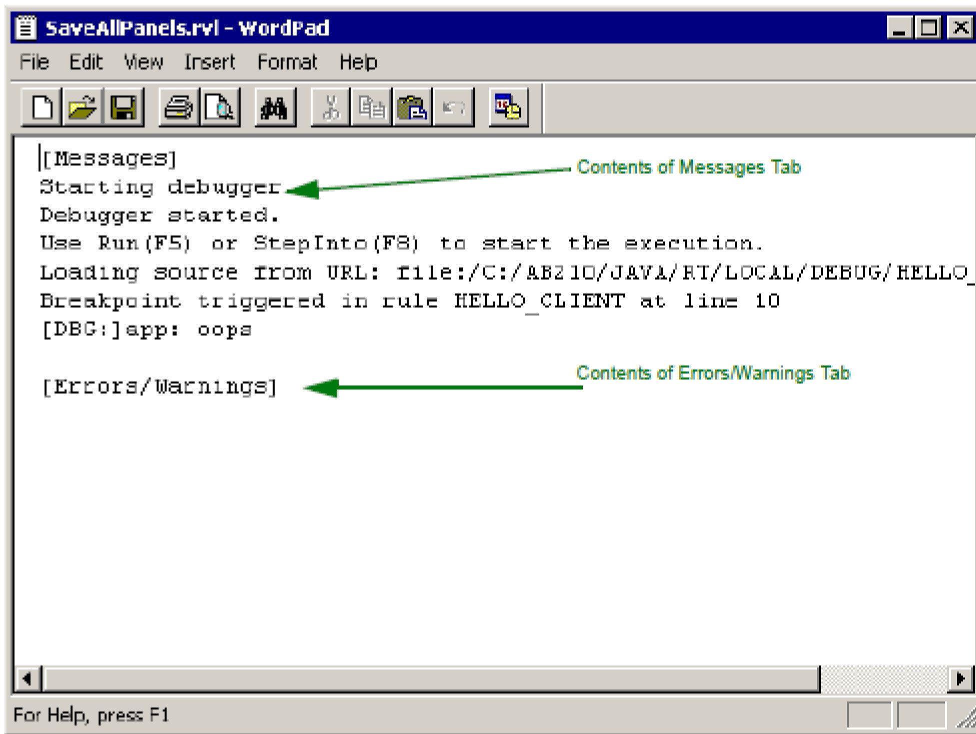
The rule source code listing window displays the Rules Language source code for rules that are running. This area shows the rule currently selected either by the execution flow, the rule hierarchy window, or the Rule Stack combo box selection. The position of the currently selected line is highlighted green. The execution pointer marks the line that is executed next for that rule, in the context of a given thread. Refer to RuleView for Java window for the location of the rule source code listing window.
Use the rule source code listing window to set breakpoints, perform text searches, and monitor the execution of the current rule.

Output Panel

The output panel reports on different events that have occurred during the debug session. It displays a rule trace and informational messages for example, that the rule source is being loaded.
There are two tabs in this panel Messages and Errors/Warnings. The Messages tab is selected by default and displays informational messages. The Errors/Warnings tab displays messages describing any error situation, such as when rule source code is not found and thus cannot be loaded. When such an error occurs, RuleView automatically switches to this tab. Refer to RuleView for Java window for the location of the output panel.
Right-click the Output panel to open a pop-up menu with the following functions:

- *Clear this panel* - This clears the contents of the selected tab.
- *Clear all panels* - This clears the contents of all the tabs.
- *Save this panel* - This saves the contents of the selected tab to a file (which you name and for which you specify the location). The default location is the AppBuilder Java debug directory.
- *Save all panels* - This saves the contents of all the tabs to a file (which you name and for which you specify the location). The contents of each tab is saved into a separate section, delimited with the name of the tag in brackets (for example, [Messages]).
- *Copy Selection* - This copies the text from the Output panel to an external text file.

Example of all tabs saved



Status Bar

The status bar contains the following items. Refer to RuleView for Java window for the location of the status bar.

- *Status* tells the status of the RuleView debugger. Refer to RuleView Status.
- *Rule Name* is the long name of the current rule (the rule being displayed).
- *Line Number* is the currently selected line number in the source code window.

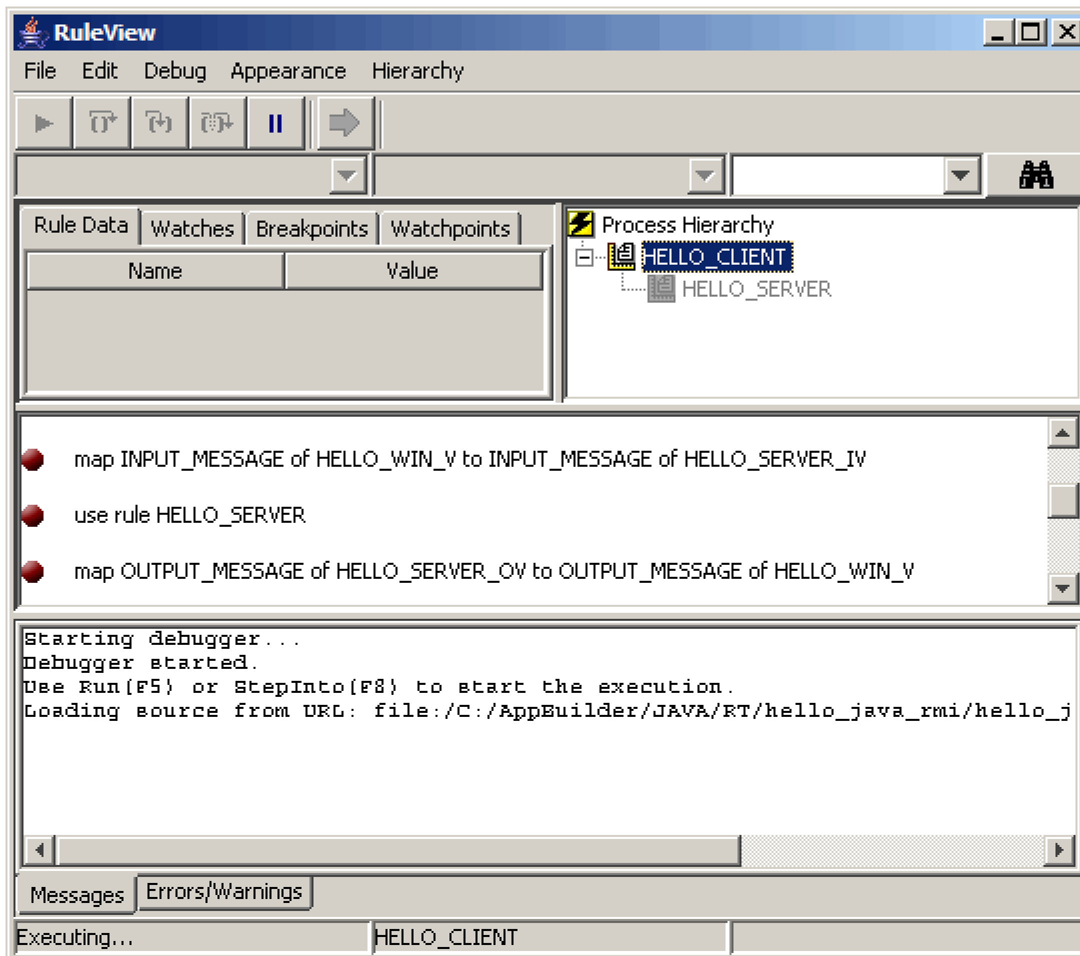
RuleView Status

Status	Description
Ready	The application is stopped and RuleView is ready to accept commands from the user.
Starting	The RuleView application is starting.
Executing	The rule being debugged is running.
Finished	The RuleView application is finished.

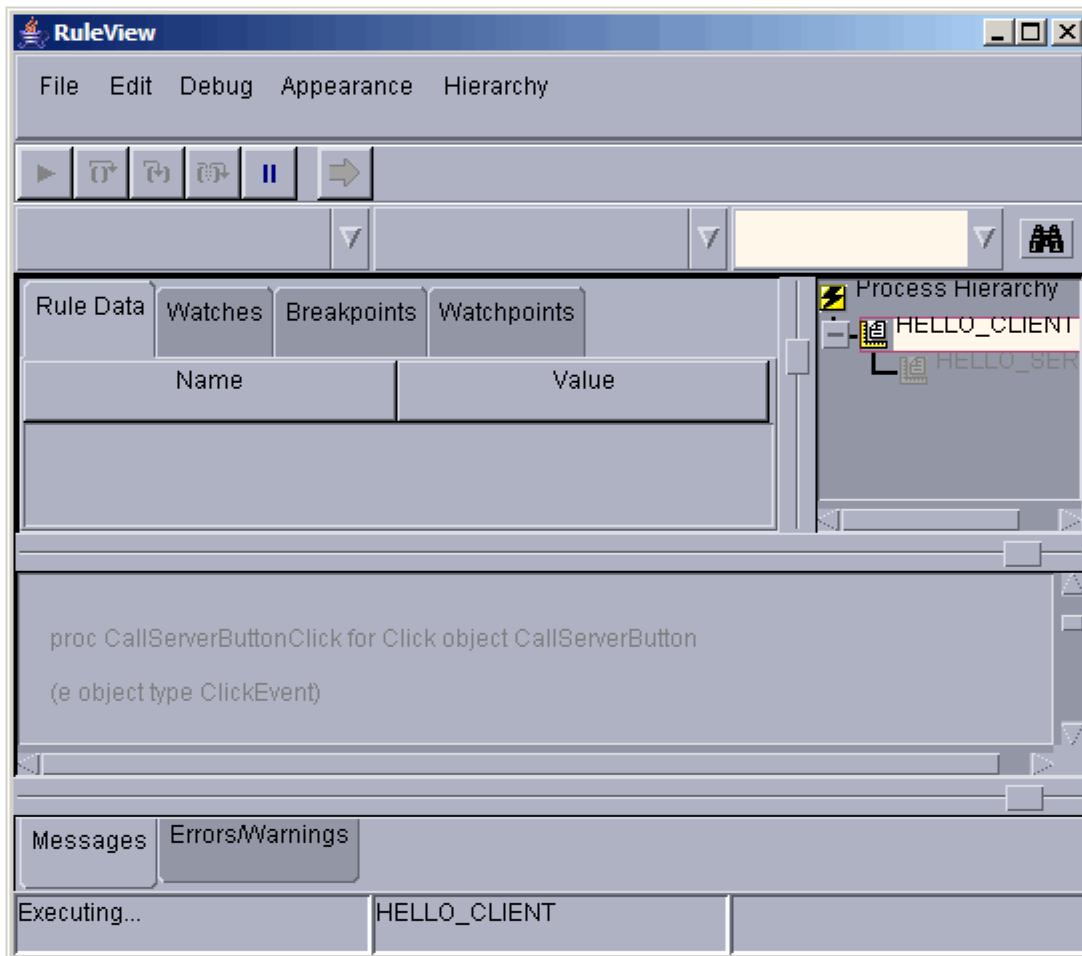
Look and Feel

You can change the look and feel of the user interface for the RuleView debugger for the Java version of the RuleView debugger only. Select from the following choices in the Appearance pull-down menu:

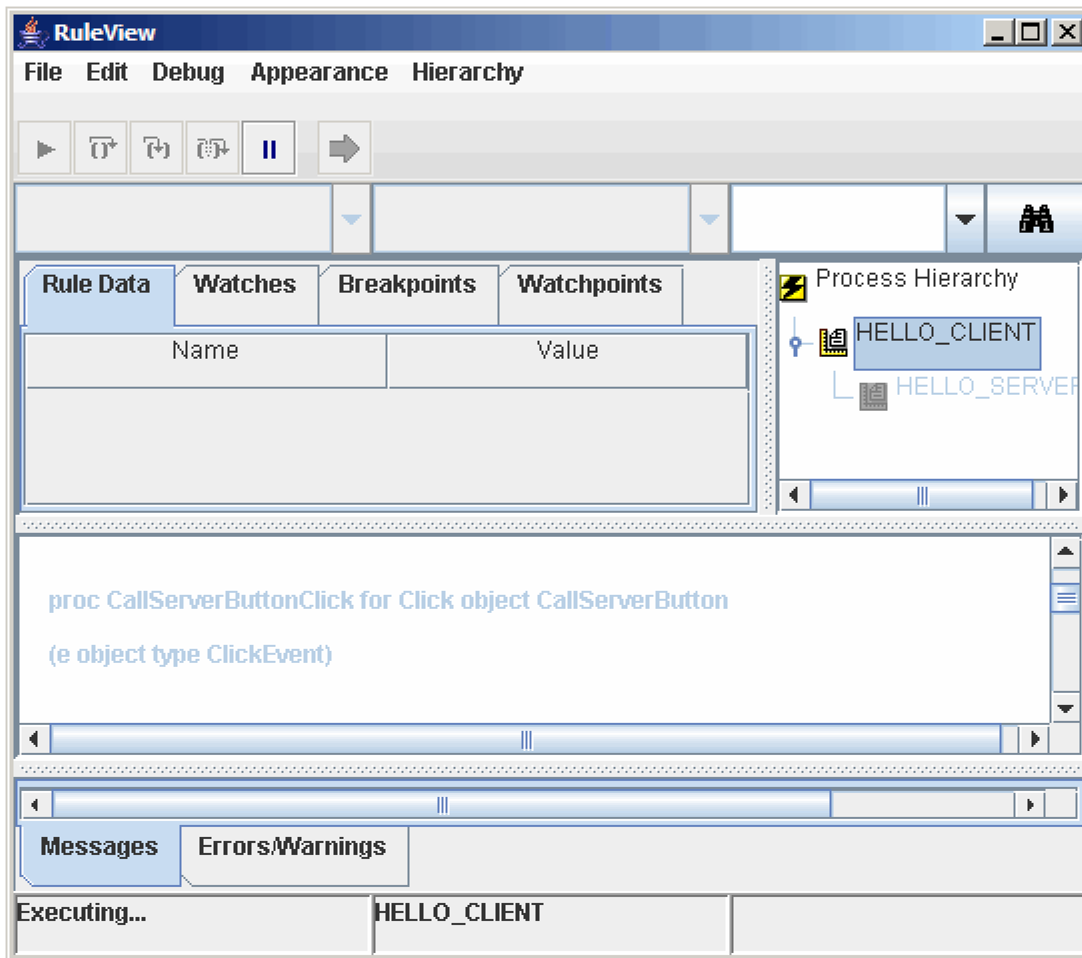
Windows (default)



Common Desktop Environment (CDE)/Motif



Metal



Depending upon the settings of your system, the Windows and Windows Classic may appear the same.

RuleView Actions

For a summary of the debug actions that can be performed from RuleView for Java, see RuleView for Java action summary.

RuleView for Java action summary

Action	Menu	Command	Toolbar Button	Accelerator Button
Continue execution	Debug	Go	Run	F5
Pause execution	Debug	Pause	Pause	
Find current location	Debug	Current Location	Current Location	F3
Trace into	Debug	Step into	Trace into	F8
Step over	Debug	Step over	Step over	F7
Step out	Debug	Step out	Step out	F6
Set breakpoint	Debug	Set breakpoint	Set breakpoint	F9
Clear breakpoint	Debug	Clear breakpoint	Clear breakpoint	F9
Enable breakpoint	Debug	Enable breakpoint		Alt+F9
Disable breakpoint	Debug	Disable breakpoint		Alt+F9
Break at Exceptions	Debug	Exceptions...		
Set watchpoint	Debug and right-click	Set Watchpoint	Set watchpoint	

Remove watchpoint	Right-click	Remove Watchpoint	Clear watchpoint	
Clear all watchpoints	Debug	Clear watchpoint		
Add watches	Right-click	Add Watch		
Remove watches	Right-click	Remove Watch		.
Modify value	Right-click	Modify value		
Set variable to null	Right-click	Set to NULL		
Add rule to hierarchy	Hierarchy	Add Rule		
Remove rule from hierarchy	Hierarchy	Remove Rule		
Perform Search	Edit	Search		Alt+S
Clear Search	Edit	Clear		Alt+X
Change look-and-feel	Appearance	LookAndFeel		
Save Settings	File	Save Settings		
Exit application	File	Exit application		Alt+F4
Exit RuleView	File	Exit RuleView		Ctrl+F4

The Set/Clear/Enable or Disable breakpoint buttons on the toolbar appear only when the action is applicable to the currently selected line of rule source code. For example, if a breakpoint is set to the currently selected line, the Set breakpoint button will not appear on the toolbar.

Running RuleView

Running RuleView

After creating, verifying, and preparing rules, debug the application using RuleView. To set RuleView to run automatically, make sure that the settings are correct in the Construction Workbench, as described in Saving and Restoring Settings. When you are ready to execute the program in debug mode, refer to Saving and Restoring Settings. To exit the debugger or the application, refer to Exiting RuleView.
This section discusses the following RuleView tasks:

- Debugging Rules with RuleView
- Pausing Execution
- Viewing and Modifying Variables (of any field within a view)
- Data Actions
- Saving and Restoring a View
- Using Breakpoints
- Using a Watchpoint
- Stepping (one statement at a time)
- Handling Exceptions
- Processing Multiple Threads
- Saving and Restoring Settings
- Initialization Settings
- Exiting RuleView
- Troubleshooting

Debugging Rules with RuleView

You can debug individual rules locally, standalone Java applications with RuleView, or a distributed Java application. For distributed applications, as long as RuleView is running in the Execution Client and the debug environment is configured correctly, RuleView handles remote calls automatically. To know where you are in the application, use Showing the Current Location.

Starting the Application

When RuleView starts, it displays its window and suspends the application just before the main rule initialization. This allows setting breakpoints (but not watchpoints). Threads, call stack, and data are not available at this time because the application has not been started yet. You can resume the application in two ways - continue the execution (see Resuming) or issue the trace into command (see Stepping Into), which stops the application again before the first rule line is executed but after the rule initialization so that the threads, call stack, and rule data are available.
Refer to Saving and Restoring Settings.

Navigating to the Rules

The rule source code listing area shows the source code of the rule. The position of the currently selected line is highlighted green in RuleView for Java. The execution pointer marks the line that is executed next for that rule, in the context of a given thread.
To find a particular part of the code in this window, use the search engine. You can use it to find a particular string of text in any line of code in that rule.

Showing the Current Location

To make it easier to find the exact source code line that is executed next, you can request the current location of the executed code in RuleView. Use *Debug* > *Show Current Location* menu or press *F3* . This brings you back to the rule and call stack frame displaying the rule source line that is to be executed next. This is performed for the currently selected thread, and the threads combo box is not affected by this command.

Resuming

The *Go* or *Run* command (*F5* , *Go* toolbar button, or *Debug* > *Go* menu item) resumes the previously suspended application. For Java, all Java virtual machine (JVM) threads are resumed until a breakpoint is reached, a watchpoint is triggered, or an exception is thrown. RuleView prints an informational message when a watchpoint is triggered and displays a message box when an exception is thrown. While the application or rule executes, only the pause command or set and remove breakpoint commands remain active.

Pausing Execution

To stop the executing of a rule without setting a breakpoint, you can pause the execution. To pause means to stop execution of the rule being debugged when one of the threads reaches the next executable line in the rule. To pause, press the *Pause* toolbar button or select *Debug Thread > Pause* from the pull-down menu. This is usually helpful to stop execution in an infinite (or extremely long) loop or in an event handler in a GUI application.
When you are done pausing, you can tell the debugger to continue the execution. To continue, press *F5* or press the *Run* toolbar button or select *Debug* > *Go* from the pull-down menu.

Viewing and Modifying Variables

RuleView displays visible variables of all types (except OBJECT, unless it is included in a VIEW) for a current stack frame in view hierarchy. Refer to RuleView for Java window for the location of the view hierarchy. The view hierarchy window contains several types of debugging data:

- Rule Data
- Watches Data

This data might change dynamically as the rule data are involved in dynamically set view operations (such as append, insert, delete, etc.).

Working with Occurring Views

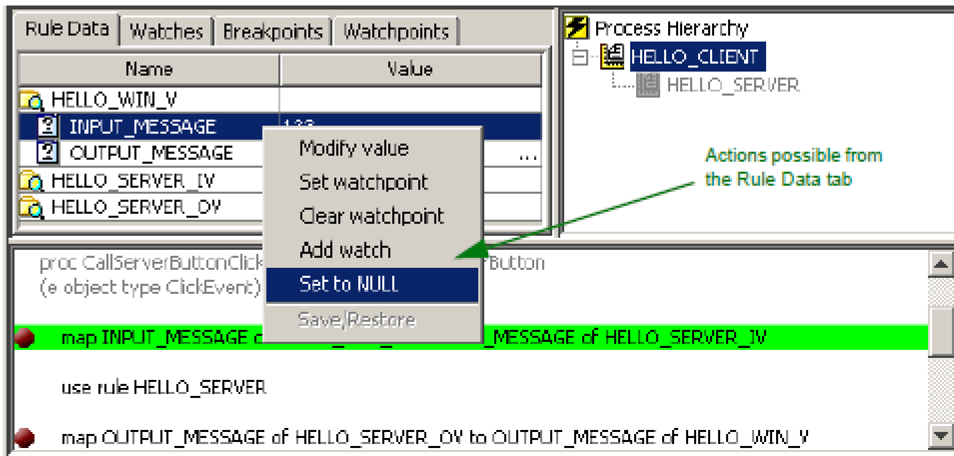
For Rule data, changes to an occurring view, if it is opened, changes the number of its elements. If any element of an occurring view is expanded, then exactly this element remains expanded unless it is deleted. For example, if the element number 4 is expanded and elements 1 through 3 are deleted, then a new element is inserted before the first element. The result is that element number 4 becomes the second element and remains expanded.
In the Watches tab, any watch (a variable being watched) is associated with the sequence if the indices are used to access the variable. Thus the contents of a watch change as a dynamically set view operation is performed. If after any sequence of operations there is no such element, the watch is deleted.

Rule Data

The Rule Data helps to browse all data accessible from the current context. The Rule Data tab consists of a table that displays the accessible rule data according to the view (input view or output view for example) to which the data belongs. These include rule global variables (including input and output views), current procedure local variables, and current rule local variables unless they are hidden by procedure data. If the current context changes (for example, if you have switched to another call stack frame), this table is updated using the new context.
Data displayed in the Rule Data panel are ordered. The following table shows the order:

- rule input view
- rule output view
- rule window view
- input view and output view of sub-rules
- input view and output view of components
- views of all sections of reports attached to the rule
- rule global views (sorted alphabetically)
- rule work views (sorted alphabetically)
- rule and procedure data, declared in DCL section (sorted alphabetically + views go before fields)

Rule Data actions



The table consists of two columns. The *Name* column displays the variable name using different icons for views and fields and reflects the structure of the data hierarchy. The *Value* column displays the current value of this variable. You can right-click any value in the data to get a pop-up menu. The following actions are allowed:

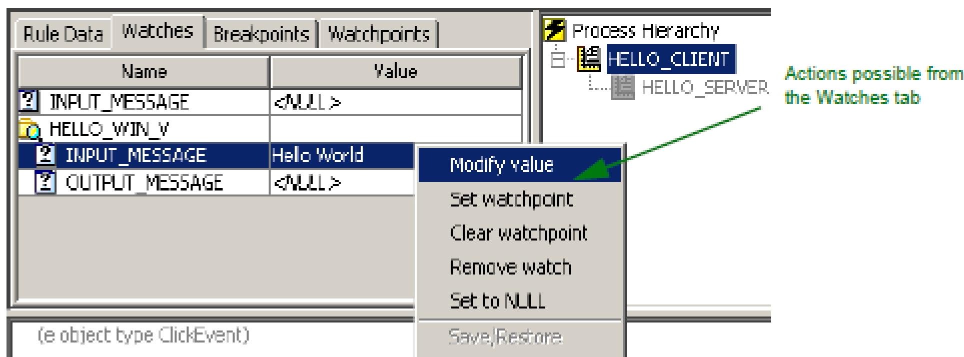
- Modify Value
- Setting a Watchpoint
- Clearing a Watchpoint
- Add Watch
- Set to NULL
- Backup (see Saving and Restoring a View)

To modify the value, the display changes, so that you can edit the value in the window. To add a watch (add a variable to watch), the Watches tab appears so that you can add a new watch.
Some of these actions may not be allowed for particular variables. For example, you cannot modify a value or set a watchpoint for a view, so these menu items are grayed out. Also, views can be saved or restored, but fields cannot.

Watches Data

The Watches data contains only variables that have been selected. The Watches tab consists of a table displaying only data selected. There are two columns of data as there are in the Rule Data Tab. The list of variables is not changed by the current context. If a variable became inaccessible from the current context, it is still displayed but with a special value, *<variable is not in scope>* . A watch that is not in scope becomes visible if an appropriate context becomes current.

Watches data actions



On the Watches data tab you can right-click any value in the data and get a pop-up menu with the following choices:

- Modify Value
- Setting a Watchpoint
- Clearing a Watchpoint
- Remove Watch
- Set to NULL
- Backup (see Saving and Restoring a View)

Data Actions

In either the Rule Data tab or the Watches tab or both, you can modify variable data. The actions are available from the pop-up menu from inside

the tables on these tabs. Each action is described in turn.

Modify Value

There are two ways to modify the value of a field in either the Rule Data tab or the Watches tab.

- Right-click the line and select *Modify value* in the pop-up menu.
- Left click the specific variable in the *Value* column.

If this variable can be edited (if it is a field and is in the scope), then the cell in the table becomes editable, and a value can be typed or edited. After editing is complete, press *Enter* to commit the result or select another variable. If editing is not complete and the debugger state has changed, then editing is canceled.

If an edited value is incorrect (for example, if you entered a non-digit for any numeric type or a date and time value that does not fit the format mask), then an error message appears asking whether to continue editing or to cancel. Press *Yes* to continue editing or *No* to leave the editor without committing the changes.

Set to NULL

To assign a NULL value to a field use the *Set to NULL* item in the pop-up menu. You can do this for fields in the Rule Data tab or the Watches tab. NULL is the initial value of any field.
This operation is similar to the *Clear* statement in the Rules Language. (Refer to the *Rules Language Reference Guide.*) A NULL value is also a special value that can be retrieved from a database. It is displayed as *<NULL>* and italicized to distinguish it from the same string value.

Add Watch

To add a watch to the table, right-click the *Rule Data* tab to select Add watch in the pop-up menu. A new watch is created, and the table on the *Watches* tab displays this watch.

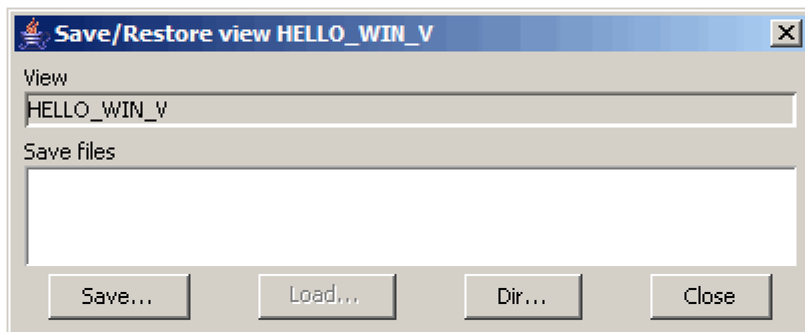
Remove Watch

To remove a watch from the table, right-click the *Watches* tab to select *Remove watch* in the pop-up menu. If the variable being watched is a view, then removing the watch removes the view along with its subviews and fields. If you remove any subview or file of a watch, it is temporarily removed until its closest containing view is collapsed and expanded again.

Saving and Restoring a View

To save a view to a file (for later use, possibly in another debug session and another rule) or to restore a previously saved view, click a view and select *Save/Restore* from the pop-up menu. The Save/Restore View dialog appears as shown in the figure below.

Save/Restore View dialog



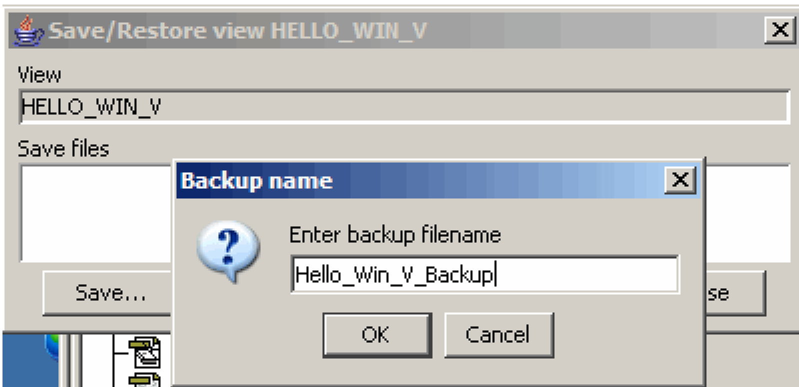
The *View* field shows the name of the view that is saved or restored and the *Save files* list shows the files containing the saved views.

Saving Views

To save a view to a file, complete the following steps:

1. Click ***Save .*** You are prompted for a file name to which the selected view is saved, as shown in Save/Restore View dialog.
2. Type a valid file name without an extension and with a valid subdirectory location and click ***OK*** .

View Backup Name dialog



You can save any view (not just the top-most ones), but you cannot save a single field. Views are saved with all fields and subviews.
If a field contains an invalid value, it is saved as a null value with a message printed to the Output Panel. (Refer to the Output Panel.)

Restoring Views

To restore a view from a file, choose one of the saved views in the *Backup files* list and click *Load* . If the topmost view that this file holds differs from those you are restoring, a *View was not saved to this file* dialog appears, and the view is not restored. While the view is saved to a file along with all its subviews, you cannot restore one of the subviews from this file; you can only restore the whole topmost view.
If a view was changed since the last save (for example, if its fields changed name or type), RuleView displays the dialog *View has been changed since last save. Try to restore same named fields?* . There are two options: you may either allow RuleView to restore the changed view partially (at your own risk) or do not restore the view. When restoring views partially, those fields that are saved and present in the current view are restored and those that are not saved in this file are reset to null. If a field has changed its type and the saved value cannot be assigned to a new field (for example, field had type CHAR(10) and became TIMESTAMP), an error message is printed to the Output Panel and such fields' value are set to null. (Refer to the Output Panel.)

Using Breakpoints

You can define a *breakpoint* in RuleView to break (or halt) the execution of the application at a specific line in your source code. When you set a breakpoint on a line and then let the application run, RuleView automatically regains control just before the breakpoint line is executed, enabling you to examine the contents of relevant views or change the values of fields. When you set a breakpoint at a specified location in the source code window, once any thread of the application reaches this location, the execution is suspended and this location is highlighted green. To run the application up to the next breakpoint, select *Debug* > *Go* , click the icon on the toolbar, or press *F5* . The Breakpoints tab provides the following operations:

- Setting a Breakpoint
- Clearing a Breakpoint
- Enable or Disable a Breakpoint
- Viewing Breakpoints

This feature is available both in suspended and unsuspended modes. Breakpoints remain active until they are removed by the Clear breakpoint command and they apply to any command. That is, if the execution reaches a breakpoint with the Step over command, execution is suspended and step over is automatically canceled.

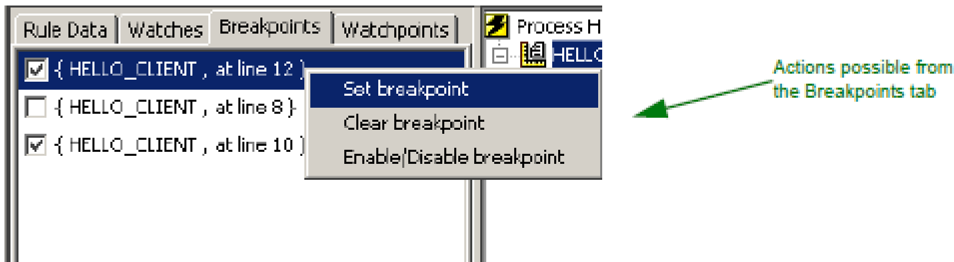
Some Stepping applies to breakpoints.

Breakpoints Tab

The Breakpoints tab consists of a checklist of breakpoints that can be enabled or disabled. You can right-click any breakpoint and get a pop-up menu. The actions that are allowed are:

- Setting a Breakpoint
- Clearing a Breakpoint
- Enable or Disable a Breakpoint

Breakpoints data actions



Setting a Breakpoint

This choice sets a breakpoint at the currently highlighted location in the source code window or at specified line. This means that when any thread of the application being debugged reaches this location, execution is suspended, and this location is highlighted in green. This command is available both in suspended and unsuspended modes.
Breakpoints can be set on the source code lines that are not grayed. However, while a rule is not loaded, the entire source is grayed, but it is possible to set a breakpoint on any line and no control on such breakpoints is performed until the rule loads. If some breakpoints are positioned on invalid lines, they are moved automatically to the next executable line. If there is no line available, the breakpoint is removed. This applies to every breakpoint (restored from the previous session or currently set). If you try to set a breakpoint (by double-clicking the rule source line, for example) on a line that is not executable and there are no executable lines below, a message is printed to the Output Panel stating that such a breakpoint cannot be set.
To set a breakpoint with RuleView for Java, do any of the following:

- With the rule displayed in the rule source window, double-click the line of code on which to set a breakpoint. (This is a toggle, so if you do it again, it disables and then clears the breakpoint.)

The line of code shows a red dot in the left margin, indicating that an active breakpoint has been added. An entry is added to the Breakpoints tab with its check box checked.

- With the rule displayed in the rule source window, click the *Set breakpoint* toolbar button, press *F9* , or select *Debug* > *Set breakpoint* from the pull-down menu.

The line of code shows a red dot in the left margin, indicating an active breakpoint has been added. An entry is added to the Breakpoints tab with its check box checked.

- In the Breakpoints tab, right-click and select *Set breakpoint* from the pop-up menu. The Adding a Breakpoint appears. Fill it out and click *OK* .

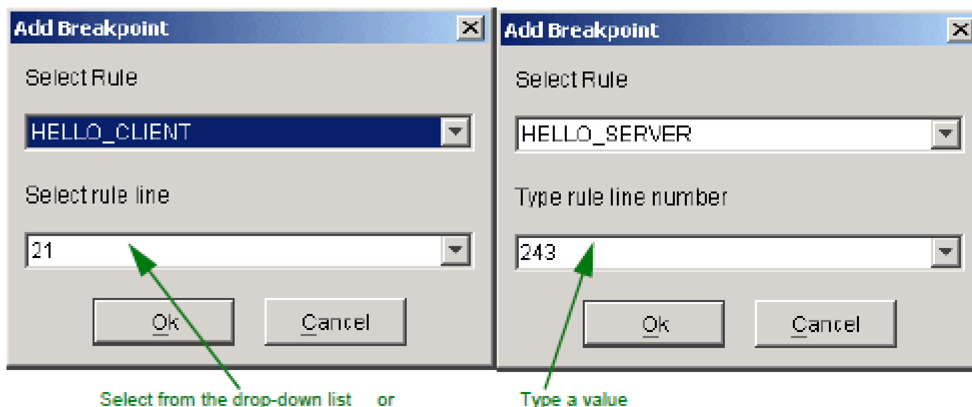
The line of code shows a red dot in the left margin, indicating that an active breakpoint has been added. An entry is added to the Breakpoints tab with its check box checked.
Some Stepping apply to breakpoints. To continue running the application, select *Debug* > *Run* from the RuleView menu. To turn off the breakpoint but leave it there for later use, refer to Enable or Disable a Breakpoint.

Adding a Breakpoint

The Add Breakpoint dialog consists of two drop-down lists. To add a breakpoint, follow the steps:

1. Using the first drop-down list, select the rule in which to set the breakpoint.
2. Select a line number in the second combo box. An example is shown in Add breakpoint dialog.

Add breakpoint dialog



The Rule drop-down list displays the accessible rules. This includes any rule that has debugging information (whether the rule is loaded or not

loaded). During execution the contents of this list might change. For instance, a rule is not loaded and so is included in the list. After this rule is loaded, it turns out that it is compiled with the no-debug-information option. From this moment, it does not appear in the list.
The next step is to choose a target rule line number. If you select a rule that is loaded, you can select an executable line number from the Rule drop-down list. For loaded rules, you may only select an executable line number from a list. If you select a rule that is not loaded, you can type the target rule line number in an editable field. There are two types of error messages concerning this input method:

- Invalid format message - this occurs when a non-integer value is entered
- Invalid line number - if the value is less than 1 or greater than the maximum line number

When done, click ***OK*** .

Enable or Disable a Breakpoint

When a breakpoint is set to a line in a rule, it can be disabled and therefore ignored by the debugger. With this feature, you can set breakpoints and disable them for some debugging activity and enable them again later. The status (of whether a breakpoint is enabled) is a toggle. There are several ways to enable or disable a breakpoint.

- With the rule displayed in the rule source window, press *Alt-F9* or select *Debug* > *Enable/disable breakpoint* from the pull-down menu.
- In the Breakpoints tab, right-click and click the check box by the breakpoint to enable or disable. If the box is checked, the breakpoint is enabled. If the box is not checked, the breakpoint is disabled.

  This menu is not available if there is no breakpoint on the selected line.

- Double-click the source line. Repeatedly double-clicking the same line with a breakpoint disables a breakpoint, then clears the breakpoint, then adds an enabled breakpoint, and so on.

Clearing a Breakpoint

To remove a breakpoint that was previously set at the currently highlighted line in the source code window, do any of the following operations:

- With the rule displayed in the rule source window, double-click the line of code on which to clear a breakpoint. (This is a toggle, so if you do it again, it sets the breakpoint.)
- With the rule displayed in the rule source window, click the ***Clear breakpoint*** toolbar button, press ***F9*** or select ***Debug > Clear breakpoint*** from the pull-down menu.
- In the Breakpoints tab, right-click the breakpoint and select ***Clear breakpoint*** from the pop-up menu.

You can select more than one line at a time to clear several or all of the breakpoints. The system removes the red dot from the left margin, indicating that there is now no breakpoint. The entry in the Breakpoints tab is removed. This command is available both in suspended and unsuspended modes. To continue running the application, select ***Debug > Run*** or the Go button from the toolbar or press ***F5***.

Viewing Breakpoints

The location of a breakpoint is indicated in the rule source code listing window by a red dot to the left of the line of code in which it is set. When a breakpoint is removed, the red dot disappears.

To see a list of the active breakpoints (including their rule long names), open the Breakpoints tab.

Restrictions

Because each breakpoint is attached to a line number rather than to a particular Rules Language statement, if you change a rule, you must verify that the points are still in the correct positions before debugging.
For example, if you set a breakpoint on line 10 and then add a line of code before line 10, the system attaches the breakpoint to the new line of code. You must manually clear the breakpoint and create a new breakpoint on line 11. Refer to Using Breakpoints for procedures on setting and clearing breakpoints.
The following Rules Language statements cannot include breakpoints or watchpoints:

- Comment lines
- Blank lines
- Any lines that include the following statements:
- CASE (but it is possible to set a breakpoint on a CASEOF statement)
- Any type of END (ENDDO, ENDIF, and so forth)
- ELSE
- DCL
- Local variable declarations (between DCL and ENDDCL)

For more information on these statements, refer to the *Rules Language Reference Guide*. Any rule source code line that cannot include a breakpoint is displayed gray.

Using a Watchpoint

The ability to set a watchpoint is only available with RuleView for Java. A watchpoint allows control over the value of a variable (field). You can set a watchpoint on a variable of any type except OBJECT type, and the debugger suspends execution flow after each variable modification and displays a message box with the event description. Using watchpoints (regardless of how many) in the debugger might decrease your application execution speed. If you are concerned with execution speed, do not use watchpoints. The possible actions with watchpoints are:

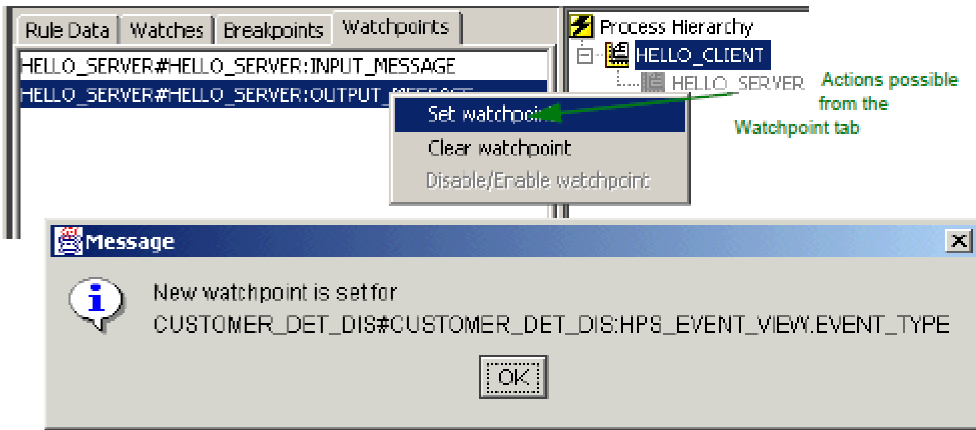
- Setting a Watchpoint
- Clearing a Watchpoint
- Viewing Watchpoints

Watchpoints Tab

The Watchpoints tab consists of a checklist of watchpoints that can be added or removed. You can right-click any watchpoint and get a pop-up menu with the following choices:

- Setting a Watchpoint
- Clearing a Watchpoint

Watchpoint data actions



Setting a Watchpoint

You can set watchpoints on any selected variable that is visible. You can set a watchpoint can be set only for fields. For views, this menu item is disabled. To set a watchpoint with RuleView for Java, do either of the following:

- With the pop-up menu in either the Rule Data tab or Watches data tab, select *Set watchpoint* .
- From the pull-down menu or Watchpoint tab, select *Debug* > *Set watchpoint* . The Add Watchpoint Dialog appears. Fill it out and click *OK* .

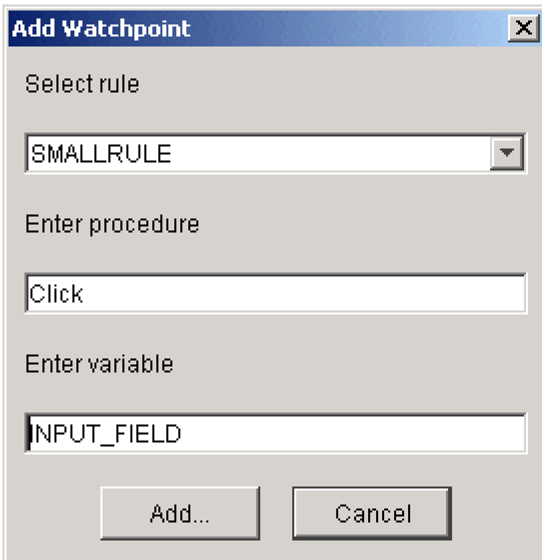
Add Watchpoint Dialog

The Select rule drop-down contains a list of currently registered rules, including the rules that have been loaded and the ones not loaded yet but does not contain loaded rules with no debug information. Select one to outline the rule data scope. Use the next two edit fields (Enter procedure and Enter variable) to select the procedure and the data object for the watchpoint.
In this Add dialog you can select a rule, then enter a procedure name and a field specification. Together, these fully represent any field in the application. A field specification that is then shown in the Watchpoints tab has the following format:

<fldSpec>::=<name>.<fldSpec> | <name>[<index>].<fldSpec> | <name>

where <name> is a name of view or field and <index> is an index of occurs view element. The field format string is case insensitive, so it does not distinguish upper or lower case letters.

Add watchpoint dialog



The possible results of this operation (printed to the Output Panel) are:

- *New watchpoint is set for <fieldSpec>* - if the watchpoint is successfully set.
- *Watchpoint for <fldSpec> already set* - if the watchpoint already exists.
- *Watchpoint could not be set* - some error occurred. This should not appear in normal operation; if you cannot find out what happened, contact Customer Support.

Press *OK* to set the watchpoint.

Example

For example, assuming we have the following view structure:

Str VARCHAR(10);

V VIEW CONTAINS Str;
Arr VIEW CONTAINS V(25);
To set a watchpoint to *STR OF V OF ARR(5)* you should type *ARR[5].V.Str* .

Clearing a Watchpoint

With the pop-up menu in either the *Rule Data* tab or the *Watches* data tab, you can remove watchpoints for selected variables. To remove a watchpoint from a field, select *Clear watchpoint* . A watchpoint can be set and removed only for fields. For views, this menu item is disabled.
To remove a watchpoint from a currently visible field, right-click to the field in the data table and select the watchpoint to remove; then, click *Clear watchpoint* from the pop-up menu.
To remove a watchpoint, switch to the *Watchpoints* tab, select the watchpoint and click *Clear watchpoint.*

Another way is to select *Debug* > *Clear Watchpoint* menu. A list of watchpoints is displayed. Select an arbitrary subset of the list and press *Remove* or *Remove All* .
Possible results of operation (printed to the Output Panel):

- *Watchpoint for <fldSpec> is removed.* - if the watchpoint was successfully removed.
- *Watchpoint for <fldSpec> is not set.* - if the watchpoint was not set.
- *Watchpoint for <fldSpec> could not be removed.* - some error occurred. This should not appear in normal operation; if you cannot find out what happened, contact Customer Support.

Viewing Watchpoints

To see a list of the active watchpoints, click the Watchpoints tab.

Stepping

You can debug rule source code one line at a time using these stepping mechanisms:

- Stepping Into
- Stepping Over
- Stepping Out

Another feature of RuleView is command buffering. It is possible to request several Trace Into, Step Over, Step Out, and Go commands by pressing the toolbar buttons or the corresponding accelerated menu items. (They are not disabled when the application you are debugging is unsuspended). They are executed in the appropriate order. If any event occurs and not all commands can be executed, you are prompted to continue execution or to discard all the commands in the queue.

Stepping Into

This command steps to the next valid (executable) rule line at any call stack level. RuleView resumes the application being debugged (as the Continue Execution command does) and suspends it when *any* thread reaches the next valid line in any rule in the hierarchy. The execution might stop in the other thread. That is, the new current thread might be other than the old one.
Use *F8* or the *Step into* toolbar button or the *Debug* > *Step into* menu to do the same.

Stepping Over

This command steps to the next valid (executable) rule line without stopping in any subrule or procedure (that is, it stops at this or the caller stack frame). Unlike trace into, step over takes into account the thread for which it is performed. A step over can be finished in that thread where it was started. If this thread is destroyed before the step finishes, the step over is cancelled automatically.
Use *F7* or the *Step over* toolbar button or the *Debug* > *Step over* menu to do the same.

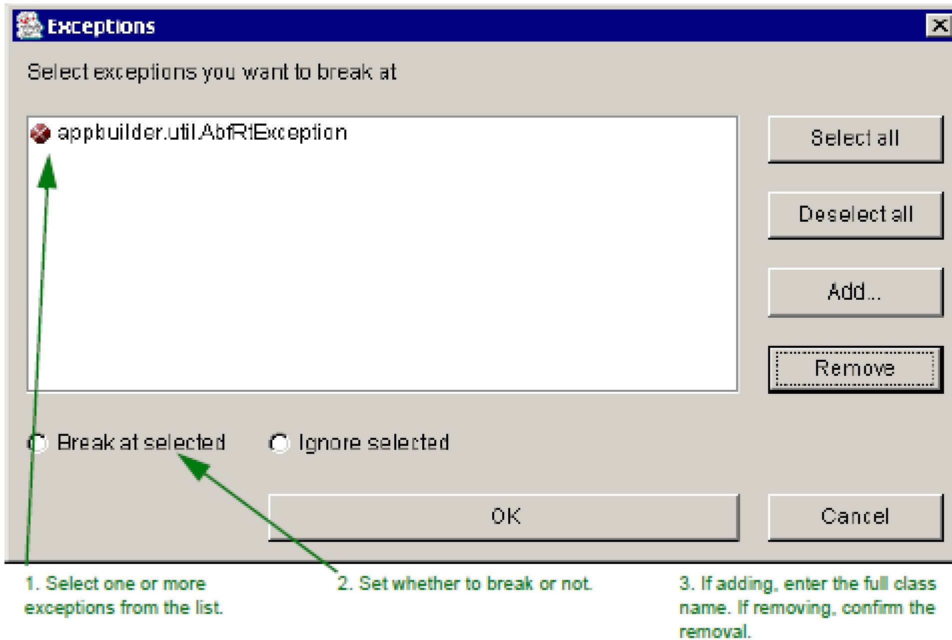
Stepping Out

This command steps out of the current subrule or procedure. More precisely, this command resumes application execution and suspends it at the next valid (executable) line in the upper stack frame of the same thread. If the thread is destroyed before it reaches the upper stack frame level, the execution is stopped at the next valid line in any thread, just as the Stepping Into command does.
Use *F6* or the *Step out* toolbar button or the *Debug* > *Step out* menu to do the same.

Handling Exceptions

AppBuilder Java applications throw AbfRtException (AppBuilder framework runtime exception) when any exception occurs in the application. Other exceptions can be added to this one, as indicated below. These exceptions are caught by the currently executing rule and then end the execution of the rule. If the rule is a root rule, then the application is also terminated.
Exception messages can be logged to the trace file or to the standard output depending on the appbuilder.ini settings APP_LEVEL and APP_FILE under the section [TRACE]. The value of APP_LEVEL must be greater than 0, and if APP_FILE is not specified the traces will go to standard output. The stack trace can also be logged when the value of appbuilder.ini setting PRINT_EXCEPTION_STACK is TRUE (in [NC] section).
RuleView can suspend an application just before an exception is thrown and display a message box containing the exception class name and exception message. It is possible to specify the exceptions at which the application is to break. You can set these in the *Exceptions* dialog (*Debug* > *Exceptions*). The content of this dialog is saved between debugging sessions (saved along with breakpoints).
To add or remove or set the handling of exceptions, select *Debug* > *Exceptions* . The Exceptions dialog appears. To add an exception to the list, click *Add* and enter the full name of the exception in the dialog. The exception class name is verified to comply with Java naming standards, but the class existence verification is not performed. To remove an exception from the list, select one or more item in the list and click *Remove* . You must confirm the removal of any exception from the dialog.
If you want to modify the spelling of the exception name, remove it and then add it back again with the correct spelling.
To set the handling of exceptions, in the Exceptions dialog, select one or more of the exceptions from the list. It is possible to use the *Select all* button or click one or more of the items in the list while holding the *Ctrl* key or *Alt* key. When you have selected the items you want to break at, click *Break at selected*. To remove any break settings you have previously set, simply select the items and click *Ignore selected*.
When you are done with adding, removing, and setting exceptions, click *OK* to continue. If you click *Cancel* , any settings you changed in this dialog are ignored except added or removed exceptions. Remember that this dialog manages only *notification* of an exceptional situation; it has no affect on exception handling. After you resume the application, the normal exception handling procedure is executed, resulting in execution-time support messages, etc.

Exceptions handling dialog



Processing Multiple Threads

RuleView for Java supports multi-threaded applications. RuleView displays only those threads that run through the rule code; if a real thread is running but not in the rule, RuleView does not display it.
For example, consider that you have a GUI application and, thus, a Java event dispatcher thread. This thread is alive until you close your application window, but the debugger shows this thread only if execution flow is stopped in some handler code (that is, in the rule). After the handler is finished, this thread is not shown, while a corresponding Java thread (the event dispatcher) is alive and still running.
RuleView does not allow suspension of one or more threads of the application without suspending the entire application. Thus, the debugging commands suspend the threads (at appropriate locations) and then enable you to inspect the stacks and the data for all threads.

Saving and Restoring Settings

RuleView can store debug session information and several user interface settings. To save settings, use *File > Save Settings Now* or *File > Save Settings on Exit* menu items.
User interface settings are somewhat global; one settings file is saved for a user if the operating system supports different home directories for different users. RuleView settings are stored in a file named ruleview.ini in the user directory. Window location, window size, and other parameters can be stored and loaded automatically upon start.
Debug session information includes breakpoints that were set during the session as well as any exceptions at which you want the program to break. Invalid, out of date breakpoints are removed automatically. While interface settings are saved only by request (Save Settings Now or Save Settings on Exit), debug session information is saved each time you exit RuleView and restored each time it is started on the same main rule.
To facilitate debugging rules in different partitions, you can set RuleView to look for debug information files in directories, specified by the DEBUG_URL setting in the TEST section of the appbuilder.ini file. If these files are not found or this setting is not specified, RuleView seeks this information in the CLASSPATH environment variable. If you are debugging a rule that belongs to some partition, append the partition directory to either the DEBUG_URL setting or the CLASSPATH.
Other auxiliary files created by RuleView (project files, saved views) are typically located in the JAVA/RT/DEBUG directory, but if the project file is not found, the folder selection dialog (titled *Choose directory to store debug files*) appears. When the directory to store the auxiliary files has been specified, the project file is saved to this directory and this directory is used to find saved views. (See Saving and Restoring a View.) Since the project file is named by the main rule long name, and this rule might appear in different partitions, choose the DEBUG directory under the partition directory to store the auxiliary debug files.

Initialization Settings

The initialization file, appbuilder.ini, contains a [DEBUG] section with several settings controlling the debugger. These parameters are summarized in Debugger initialization settings.

Debugger initialization settings

Parameter	Description
CONNECT_TIMEOUT	Timeout value in seconds is used for establishing a connection between debugger and remote server (for client/server debugging only). The default value is 30.

DBG_JVM_NAME	Specify the name of the Java virtual machine (JVM) executable to start when using RuleView. The default is java.exe. This setting is for the execution client, not for RuleView. This executable is the JVM on which to start the execution client.
DBG_JVM_PARAMETERS	Specify any additional parameters for JVM. Use *-classic* if using JDK 1.3

RuleView also uses settings from the [TEST] section of the appbuilder.ini file:

Java Debugger settings from the TEST section

Parameter	**Description**
DEBUG_START	Specify whether or not to invoke the RuleView debugger when the main rule starts. The default value of QUERY causes a dialog box to be displayed which prompts the user to choose. Valid values are TRUE, FALSE, and QUERY.
DEBUG_URL	Specify the location of the rule source and miscellaneous debug information files used by the RuleView debugger. The default is in the local classpath under /debug.
MENUFILE_URL	Specify the location of the menu file (menufile.mnu). The default is in the local classpath under /menu/menufile/mnu. This setting is used by the Execution Client.

Exiting RuleView

Follow the below instructions to exit RuleView.

Exiting the Application

To exit the application, select *File* > *Exit application* from the pull-down menu or press *Alt+F4* . This terminates the application being debugged and, for Java applications, terminates the target Java virtual machine (JVM), and exits from RuleView. Before performing this action, RuleView displays a dialog asking *Do you want to close this debugging session?* to which you can respond *Yes* or *No* .
You can also click the *X* button in the upper right corner of the RuleView window. This is the same as *Exit application* .

Exiting RuleView Only

To exit RuleView, select ***File > Exit RuleView*** from the pull-down menu or press ***Ctrl+F4***. This exits from RuleView without the application (that is being debugged) terminating. The application continues to run.

When Finished

When the application that you are debugging is terminated, use one of the following methods to exit RuleView:

- Press ***Ctrl+F4***.
- Select ***File > Exit RuleView***.
- Click the ***X*** button in the upper right corner of the RuleView window.

RuleView gives you a chance to view the application trace, if it is necessary. (For information about the trace output, refer to Output Panel).

Troubleshooting

Troubleshooting

If you are having problems with RuleView, the cause might be any of the following:

- Known Issues
- Java Interpreter Crash
- Loop Stepping Over
- Common Problems
- Local Variables Not Displayed
- Debugger Not Stepping into Subrules
- Pause Execution Command Not Working
- Debugging Rules without Debug Information
- Debug Information or Rule Source are Inconsistent with Generated Rule Code
- RuleView not Displaying Rules Properly
- Statement Triggered Watchpoint Selected with Green, not Red

Check this short list before asking for assistance.

Known Issues

The following issues are known and are being resolved.

Java Interpreter Crash

Java interpreter crashes sometimes when trying to debug a rule that has several threads. Most frequently it is caused by a pause request in RuleView.

Loop Stepping Over

RuleView steps over some loops (when an end-user is performing step over command) that have a body that consists of one line only. For example, it does not stop inside the loop.

Common Problems

This section presents the list of the common problems.

Local Variables Not Displayed

Local variables are objects or object pointers. The RuleView debugger does not support object or object pointers inspection.

Debugger Not Stepping into Subrules

Make sure that you have prepared these rules with debug information.

Pause Execution Command Not Working

Pause command stops execution flow only when it reaches a valid rule line, so if you are debugging a GUI application, the debugger is stopped only after some handler within a rule is called.

Debugging Rules without Debug Information

You can debug an application with one or several rules prepared with no debug info using RuleView. However, the absence of debug information prevents you from stopping execution in such a rule and viewing its variables.
Also, since information on subrules of such rule is not available either, you must add subrules manually or wait until these rules are executed and then they are added to the hierarchy automatically.
The source code for subrules is available anyway.

Debug Information or Rule Source are Inconsistent with Generated Rule Code

When RuleView loads the debug information of a rule, it verifies if it is consistent with the generated rule code. Debug information might become inconsistent if you, for example, copy the latest version of a generated rule on another machine but didn't take debug the information class. To inform you of such a situation, an error message is printed to the Output Panel:

Debug info inconsistent with rule. Please, reprepare this rule.

Such a rule is treated as one without debug information (see Debugging Rules without Debug Information).
Rule source consistency is also verified. If the rule source code RuleView loaded is newer or older than it should be, the following message is printed to the Output Panel:

Rule source code is inconsistent with rule debug info; locations reported might be invalid. Please, reprepare this rule.

This is only a warning message and does not prevent debugging this rule.

RuleView not Displaying Rules Properly

RuleView depends upon auxiliary files generated during preparation to locate rules used in the application. If the function being debugged has not been super prepared, or if rules have been added or removed since the last super prepare, RuleView might not be able to display the rule hierarchy properly. Consequently, you might not be able to view the source code for some rules.

Statement Triggered Watchpoint Selected with Green, not Red

When watchpoint is triggered, RuleView usually stops at the next line, highlighting a line where the watchpoint was triggered with Red. However, under certain circumstances, it might stop at the same line and in this case green overlays red. This usually happens inside loops and procedures.

Debugging .NET applications

AppBuilder 3.2 includes flexible tracing and logging capabilities. Tracing and logging can be configured in the INI file of the rule or Bphx.Sdf.ini (if

no rule-specific INI file is used). This INI file's default location is <AppBuilder>\DotNet\sys\bin; however, the INI file must be located in the same location from where you are running the application. When you open the application configuration, then the partition, the BIN folder should have the INI file in it. You can also use custom .NET logging mechanisms like log4net, if you desire.

Tracing Options

There are four tracing options.

Tracing options

Option	Possible Values	Description
UseDefaultLogger		Specifies that the default system logger will be used.
LogFilters	All None ModuleEvents ControlEvents Configuration	Everything will be logged Nothing will be logged Module related events will be logged GUI Control related events will be logged Configuration related problems will be logged
Logger levels • AppLevel • ErrLevel • SysLevel	All Info Debug Warning Error Fatal No (These settings are relevant only when used with DefaultLogger.)	Specifies what will be logged by the application logger Specifies what will be logged by error logger Specifies what will be logged by system logger
CustomLoggerConfigFile		Specifies configuration file to be used by custom logger. Usually log4net configuration file.

Getting Ready to Debug in .NET

This section explains the step by step process using Visual Studio.NET 2005 to debug AppBuilder-generated applications. You can use any version of Visual Studio.NET 2005 (including the Express Edition) for debugging purposes. There is no AppBuilder RuleView for .NET applications.

To debug an AppBuilder-generated application using VisualStudio.NET:

1. Create a folder where you want the VS.NET project to debug an application.
2. Copy the generated files to the specified folder. Copy only the .cs files.
3. Create a resource folder in the specified folder.
4. Copy all resource files into the resource folder just created.
5. Open the VS.NET 2005 IDE.
6. Go to File >New > Project from the existing code. If this option is not available, create a Console Project and copy the source file into it.
7. For the type of project, specify Visual C#.
8. Specify the location of the application folder (created in step 1 above) in the "Where are the files?" textbox.
9. Leave the output type as Console (in order to output trace and log messages).
10. Click Finish.
11. Check the project tree that is created.
12. Select all files in the resources folder (created in step 3 above), go to Properties and set Build Action to Embedded resource for all items in this folder.
13. Add a reference to Bphx.Sdf.dll.
14. For client applications add a reference to System.Drawing, System.Windows.Forms, Bphx.Sdf.dll and Bphx.Sdf.Client.dll.
15. Set an entry point for the project. The entry point is usually the root rule of the application.
16. Build the project and correct any errors that might have occurred.
17. Create or copy the existing Bphx.Sdf.ini file.
    - Create or copy this file in the bin directory or
    - Create the file in the VS.NET project and set the Build action to Content and "Copy to output directory" as "Copy always" or "Copy if newer."
18. In the GUI section of the INI file, set LoadResourcesFromAssembly option to True.
19. In the GUI section of the INI file, set DefaultResourceNamespace option to <ProjectName>.<ResourceFolder>. For example, AppBuilderProject.Resources.
20. Build and run the solution.

You can set breakpoints and debug the application as desired.

Other Debug Options

You can use other, third-party, tools to debug AppBuilder-generated applications. In general, these tools should be able to debug any .NET 2.0 application.

Debugging C Applications with RuleView

C applications can be debugged locally with AppBuilder RuleView for C. This section discusses using RuleView for C to look at rules deployed as C applications and explains the available functions in the RuleView interface for C Language application development. Before debugging the application, verify the syntax and logic of the individual rules and solve any preparation problems. The process of verification is explained in the *Developing Applications Guide* , and the troubleshooting section covers preparation problems in the *Deploying Applications Guide* . For information about RuleView for Java, refer to Debugging Java Applications.
This chapter covers the following RuleView for C topics:

- Getting Ready to Debug
- Starting RuleView for C
- Understanding the RuleView Interface
- Debugging with RuleView for C
- RuleView for C Window Settings
- Exiting RuleView for C

Getting Ready to Debug in C

Getting Ready to Debug

Before running the debugger for an application, follow these steps to set the Rule Debug option.
To enable the debug option:

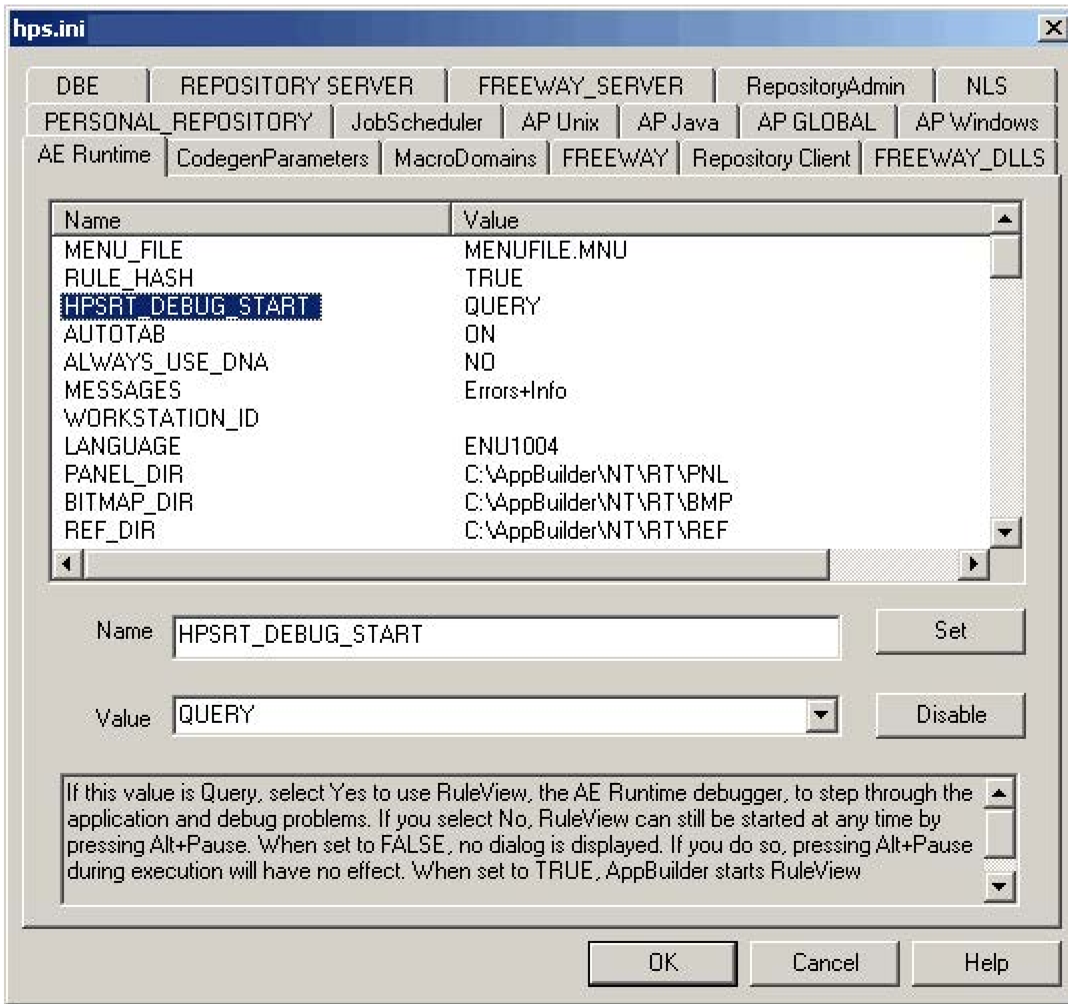
1. From Construction Workbench, select **Tools > Workbench Options**.
2. Click the **Preparation** tab.
3. Make sure that the Rule Debug box is checked.

Refer to the *Deploying Applications Guide* for more information.
You can also change the ini setting to enable the use of RuleView for debugging C applications. You can manually adjust the setting in the ini file, or use the Management Console. The ini file is located in the directory where AppBuilder is installed.

To enable the RuleView for C applications, follow the steps:

1. Click Windows Start menu and select **All Programs > AppBuilder > Configuration > Management Console**.
2. In the AppBuilder Configuration dialog, right-click your computer name and select **Edit Hps.ini** from the pop-up menu.
3. When the Hps.ini file loads, select the AE Runtime tab.

Figure 4-1 Hps.ini file [AE Runtime] section HPSRT_DEBUG_START setting



;

4. Find the HPSRT_DEBUG_START setting. Select one of the following values from the pull-down options:
   - **QUERY** – This is the default setting. The system prompts you select whether or not you want to start RuleView when the application is run.
   - **TRUE** – The system always starts RuleView when the application is run without asking.
   - **FALSE** – The system does not start RuleView when the application is run. This setting is always set on production client workstations.

The debug option setting and the RuleView start setting stay in effect regardless of how you start the application — from the Construction Workbench or outside the Construction Workbench.

Starting RuleView for C

Starting RuleView for C

To set RuleView to run automatically, make sure that the settings are correct in the Construction Workbench, as described in Debugging with RuleView for C.
To debug an application, start RuleView from the Construction Workbench menu and complete the following steps:

1. Select **Run > Windows** from the Construction Workbench menu. The Windows Execution client appears.
2. Select the function to execute. If the Debug Start Value has been set to QUERY, the system displays a dialog with the following message:
   Do you wish to start the RuleView?
3. Click **Yes**.
4. To debug an individual rule developed in C, open the rule object in Construction Workbench.
5. Right-click and select **Debug Rule** from the menu.

By default, AppBuilder prompts you to start RuleView when you run your application. You can configure AppBuilder either to use RuleView always or never by selecting the **Rule debug** option on the **Preparation** tab of the Workbench Options window. Refer to the *Development Tools Reference Guide* for more information. You can also change the *HPSRT_DEBUG_START* statement in the AE Runtime section of the *hps.ini* file. Refer to the *Communications Guide* for more instructions on editing the INI file from the Management Console.

RuleView can be started even if you select **No** from the dialog that asks if you want to run RuleView. Press **Alt+Pause** to start RuleView while the application is conversing a window. The RuleView window appears but remains inactive until you perform an action in the application window that ends the converse. Then the application stops and RuleView becomes active. This only works if you selected **No** from the dialog; you cannot start RuleView by pressing **Alt+Pause** if the value of the runtime start debug variable is set to **False** .

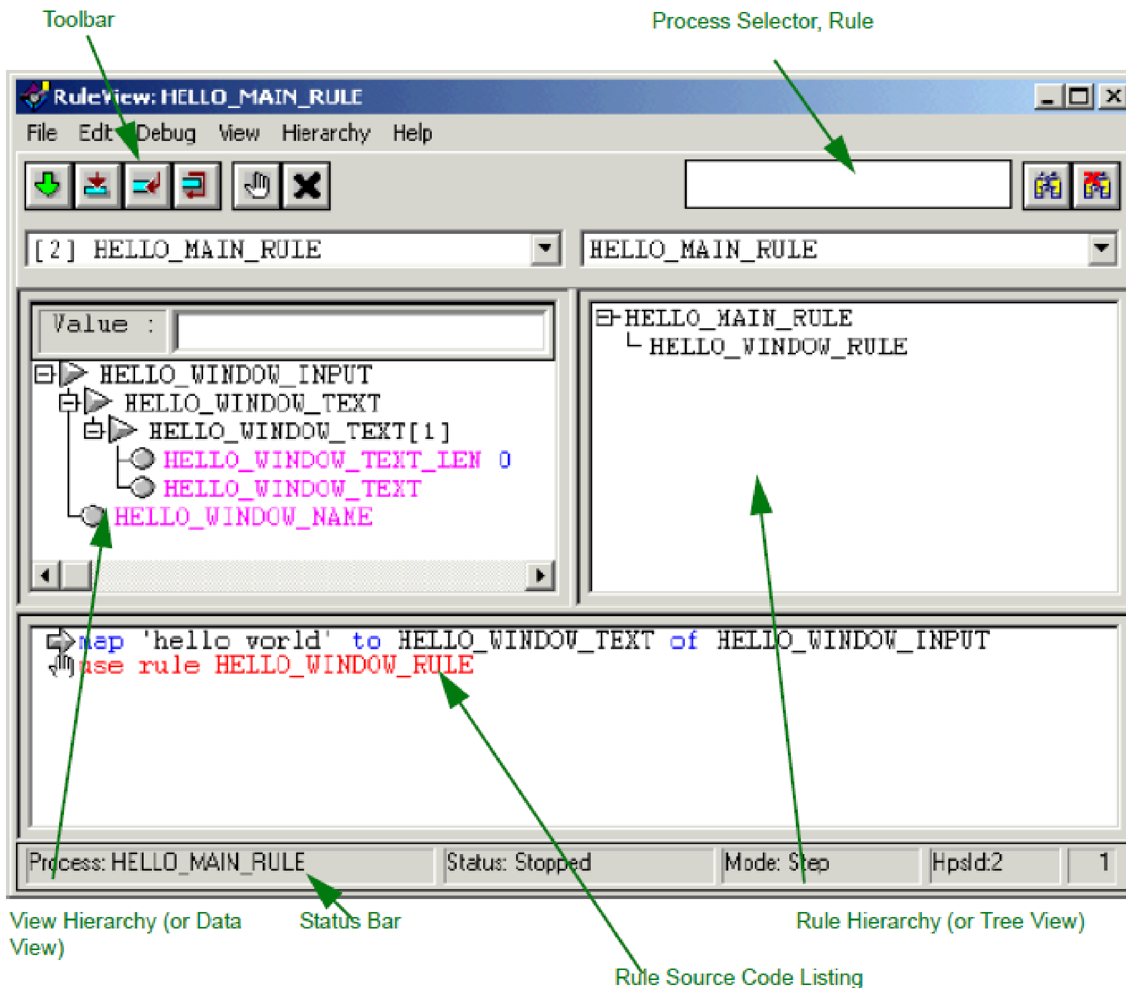
Understanding the RuleView Interface

Understanding the RuleView Interface

RuleView is a Rules Language source code debugger that has its own graphical user interface. Once you become familiar with the user interface, you can perform the debugging operations. The interface contains the following sections:

- Pull-down Menu
- Toolbar
- Process Selector, Rule Stack, and Search
- Window Areas
- Status Bar
- RuleView Actions

Figure 4-2 RuleView for C window



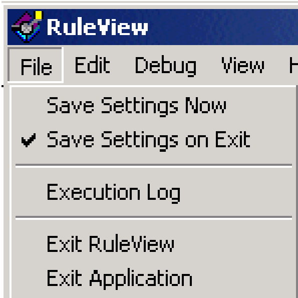
Pull-down Menu

Pull-down menus provide the controls for the debugging actions. Many of these functions are available in right-click menus and from buttons in the Toolbar. Use the RuleView toolbar to access the RuleView Actions quickly.

File menu

Use the File menu to exit the application being debugged or just exit RuleView without affecting the execution of the application.

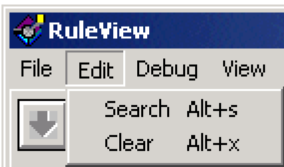
Figure 4-3 File menu - RuleView for C



Edit menu

Use the Edit menu to perform a search or to clear the current search.

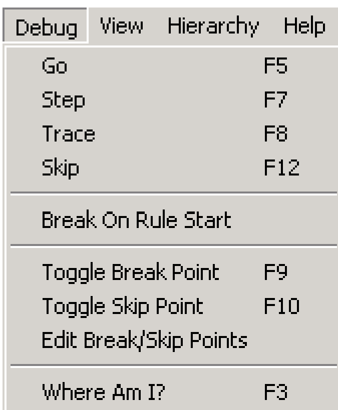
Figure 4-4 Edit menu - RuleView for C



Debug menu

The Debug menu has most of the debugging actions, from stepping through the execution to setting and clearing action points. Besides breakpoints, you can also set skippoints.

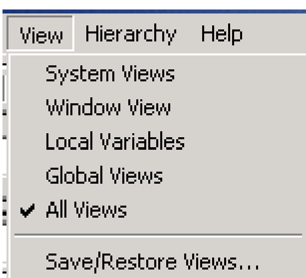
Figure 4-5 Debug menu - RuleView for C



View menu

Use the View menu to set the filter on views currently displaying.

Figure 4-6 View menu - RuleView for C



Hierarchy menu

Use the Hierarchy to set filters for objects currently displaying in the hierarchy.

Figure 4-7 Hierarchy menu - RuleView for C

Toolbar

Use the RuleView toolbar to quickly access the RuleView Actions.

Figure 4-7 RuleView for C toolbar

Process Selector, Rule Stack, and Search

When an application runs, it executes rules, beginning with its *root rule* . If the root rule uses another rule, the second rule is added to the list of rules that are currently executing, which is called the rule stack. The second rule can itself use another rule, which is added as well. The rule at the top of the stack is called the *executing rule* . When the executing rule finishes, it is removed from the stack. When a rule is selected from the rule stack combo box, the rule source code listing window displays the source code of the selected rule, and the view hierarchy window shows its associated views.

The application described in the preceding paragraph has only one process, which has the name of the root rule, and its Rule Stack holds all of the rules of the application. If the application contains a USE RULE DETACH statement, the detached rule has a separate process that RuleView displays concurrently with the root process. In other words, if you have multiple processes, multiple process contexts are displayed. In the C version, the process is assigned a system identifier (HPSID) as well as being named according to its root rule. If the same (detached) rule is used more than once, multiple instances of the process can be distinguished by their unique system numbers.

Refer to Figure 4-2 for the location of the combo boxes. The Rule Stack combo box shows all the available rules for the current thread. Choose one in the combo box to see the corresponding source code and available data. The Thread combo box shows all currently available threads. It is used to switch between threads and to view corresponding source code, available stacks, and their data. The Search window allows searching for an alphanumeric string in the rule source code. The source code search engine is a text field to the right of the Rule Stack combo box. Type the string to look for and press *Enter* . The first source line containing the string being searched is selected. To find the next occurrence of that string, press *Alt* + *S* or *Edit* > *Search* in the source code window.

Window Areas

The window areas include:

- Rule Hierarchy
- View Hierarchy
- Rule Source Code Listing

Rule Hierarchy

The rules, components, and windows associated with the application being run are shown in the Rule Hierarchy (or Tree View) window. The rules are displayed in a tree format, with each rule displayed as the child of the rule that uses it. You can browse through the view by expanding or collapsing branches of the tree. Components and windows are distinguished from rules by different icons. The Rule Hierarchy is in the upper right corner of the window. Click the tree item to see the source code corresponding to that item. Use the Hierarchy pull-down menu to select which elements are displayed in the Rule Hierarchy window. Refer to Figure 4-2 for the location of the Rule Hierarchy window.

To display the source code for a rule in the rule source window, double-click the rule in the Rule Hierarchy window or select the rule in the Rule Hierarchy window and press *Enter* .

RuleView depends upon auxiliary files generated during preparation to locate rules used in the application. If the function being debugged has not been super prepared, or if rules have been added or removed since the last super prepare, RuleView might not be able to display the rule hierarchy properly. Consequently, it might not be possible to view the source code for some rules.

View Hierarchy

This area displays the views associated with the rule currently displayed in the rule source window. If the rule being displayed is not loaded, the view hierarchy (or data view) window is blank. This is in the upper left corner of the RuleView window. It displays all the data available in the current stack. Expand non-leaf nodes to see the value of the fields of previously selected views. This is a tree display so that you can browse through the view by expanding or collapsing branches of the tree. Refer to Figure 4-2 for the location of the view hierarchy window.

Rule Source Code Listing

The rule source code listing window displays the Rules Language source code for rules that are being executed. This area shows the rule currently selected either by the execution flow, the Rule Hierarchy window, or the Rule Stack combo box selection. The position of the currently selected line is indicated by an arrow (called the execution pointer) in the RuleView interface. The execution pointer marks the line that is executed next for that rule, in the context of a given process. Refer to Figure 4-2 for the location of the rule source code listing window.

Use the rule source window to set breakpoints and skippoints, perform text searches, and monitor execution of the current rule. Color is used to highlight comments, keywords, symbols, and breakpoints. Icons on the left edge of the rule source window identify source line properties. A colored-bar cursor indicates the selected line within the source code and is used to specify the target line for adjusting breakpoints and skippoints. Additionally, the cursor indicates the progress of text searches.

Status Bar

The status bar at the bottom of the window displays relevant information about the state of the execution of the rule or application. The status bar identifies the following items.

- *Process* identifies the process context displayed in the window.
- *Status* tells whether the application is stopped (and RuleView is active) or running (and RuleView is inactive).
- *Mode* is the way RuleView is executing the process: step (stop at each line) or go (run until it hits a breakpoint).
- *System ID* is the unique number that differentiates multiple instances of a process. This number is also displayed in brackets before the process name in the process selector on the toolbar.

Refer to Figure 4-2 for the location of the status bar.

RuleView Actions

For a summary of the debug actions that can be performed from RuleView for C, see Table 4-1.

Table 4-1 RuleView for C action summary

Action	Menu	Command	Toolbar Button	Accelerator Button
Continue execution	Debug	Go	Run	F5
Pause execution	Debug	Pause	Pause	
Step over	Debug	Step over	Step over	F7
Step into	Debug	Step into	Step into	F8
Step out	Debug	Step out	Step out	F6
Set breakpoint	Debug	Set breakpoint	Set breakpoint	F9
Clear breakpoint	Debug	Clear breakpoint	Clear breakpoint	
Set watchpoint	Right-click	Set Watchpoint		
Remove watchpoint	Right-click	Remove Watchpoint		
Clear all watchpoints	Debug	Clear watchpoint		
Exit application	File	Exit application		Alt+F4
Exit RuleView	File	Exit rule view		Ctrl+F4
Add or remove watches	Right-click	Add Watch		

Modify value	Right-click	Modify value		.

Debugging with RuleView for C

Debugging with RuleView for C

You can debug individual rules locally or standalone C applications with RuleView.
With RuleView, you can perform the following debugging tasks:

- Finding and Viewing Rule Source (and any rule in its data universe)
- Editing Contents of a Value (of any field within a view)
- Stepping Through a Rule (one statement at a time)
- Tracing the Execution
- Using Action Points (to break on any line of rule source code or skip a step)

For a quick summary of the debug functions that can be performed and the part of the RuleView interface that is used to perform that action, see Table 4-1.

Finding and Viewing Rule Source

The rule source code listing area shows the source code of the rule. The position of the currently selected line is indicated by an arrow (called the execution pointer) in the RuleView interface. The execution pointer marks the line that is executed next for that rule, in the context of a given process. By default, the Rule Hierarchy, View Hierarchy, and Rule Source Code Listing windows open when you run RuleView. You can still change the look and feel of the rule source; refer to RuleView for C Window Settings.

To find a particular part of the code in this window, use the Process Selector, Rule Stack, and Search facilities in the toolbar. The search engine helps to find a particular string of text in any line of code in that rule.

Editing Contents of a Value

Use the view hierarchy Value edit control above the View Hierarchy to enter or modify data for fields (variables) used by the displayed rule. When you click a field node within the tree, the value of the field is displayed in the Value edit control.

To edit the value of a field, select the field and click the Value edit control and double-click the field or select the field and type. To commit the change, move the keyboard focus out of the Value edit field (for example, select another field node or click on a different part of the window) or press **Enter** . To abandon the change, press **Esc** . If you continue execution while editing a field value, the changes are cancelled automatically.

Stepping Through a Rule

To execute each line of code in the rule, one line or step at a time, RuleView allows stepping through the rule. After the application executes that line, control returns to the RuleView. To step through the rule, select **Debug > Step** from the pull-down menu or press **F7** or click **Step** button in the toolbar. Select **Step** again to execute the next executable line of the rule. Continue stepping through the entire rule.

If RuleView encounters a *USE RULE* statement, the system executes both the "using" and "used" rules before returning to the RuleView. You must trace into the rule in order to step through the second rule. See Tracing the Execution.

Trace (or step into) steps to the next executable rule line and enters a procedure or subrule. A subrule is the child rule of a parent rule – a rule called by another rule. Since the debugger is following the flow of execution of a thread or process, it may follow the execution of individual rules that are called by the initial rule being debugged. Use **F8** or **Trace** toolbar button or **Debug > Trace** menu to do the same.
Step steps to the next executable rule line without stopping in any subrules or procedures. That is, it stops at this or the calling stack. Use **F7** or **Step** toolbar button or **Debug > Step** menu to do the same.

Tracing the Execution

It is possible to specify that the trace information print to a file. In the *AppBuilder* \ne20tcp directory, in the *dna.ini* file, find the *TRACING* section. In the TRCFILE variable, type the name of the output file for traces. It is also possible to specify the maximum file size and the name and location of a backup file for the previous trace.
The default values for that section are:

```
[TRACING]
DEBUGLVL=ERRORS
TRCFILE=C:\AppBuilder\netetcp\trace.out
HPSCATS=0:err.cat, 8:hdb.cat
HPSCATSFILES=C:\AppBuilder\MSG
LANG=C
TRACE_FILE_SIZE=1M
BACKUP_FILE_NAME=C:\AppBuilder\netetcp\trace.bak
DUMPDATA=INOUT
DUMP_INPUT_SIZE=512
DUMP_OUTPUT_SIZE=512
```

Using Action Points

There are action points that stop or skip or evaluate variables at certain points (or lines of code). These include:

- Setting and Clearing a Breakpoint
- Setting and Clearing a Skippoint

You can also customize the RuleView window by using specifying the settings in an editable file. See RuleView for C Window Settings.

Setting and Clearing a Breakpoint

RuleView allows breaking (or halting) the execution of the application at a specific line in the source code, called a breakpoint. When you set a breakpoint on a line and then let the application run, RuleView automatically regains control just before the breakpoint line is executed, enabling you to examine the contents of relevant views or change the values of fields. Once you set a breakpoint at the selected location in the source code window, when any thread of the application (or rule) reaches this location, execution is suspended and has the pointer. To run the application up to the next breakpoint, select **Debug > Go** , click the icon on the toolbar, or press **F5** .

This feature is available both in suspended and unsuspended modes. Breakpoints remain active until they are removed by the clear breakpoint command and they affect any command. That is, if the execution reaches a breakpoint with the step over command, execution is suspended and step over is automatically canceled.

Some restrictions apply to breakpoins. For more information, refer to Restrictions.

Set a Breakpoint

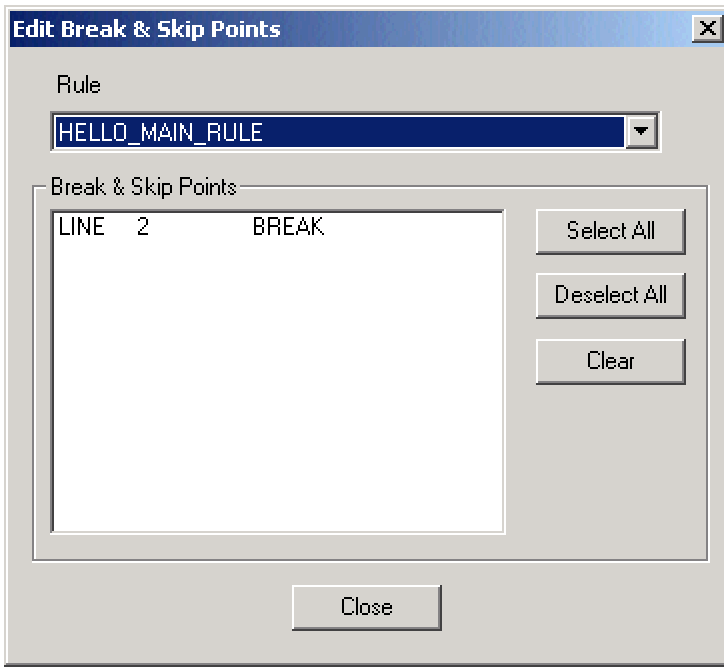
To set a breakpoint with RuleView:

1. With the rule displayed in the rule source window, select the line of code on which to set a breakpoint.
2. Select ***Debug > Toggle Breakpoint*** from the RuleView menu, then double-click the selected line. It is also possible to set a breakpoint by clicking on the **Breakpoint** icon in the toolbar or pressing **F5** .
   The system displays the breakpoint icon on the line.
3. To continue running the application, select **Debug > Run** from the RuleView menu.

To edit the breakpoints:

1. Select **Debug > Edit Break/Skip Points** from the RuleView menu.

   Figure 4-9 Edit breakpoints and skippoints dialog



2. Select the breakpoints for any rule in the Rule drop-down list. Click **Close** when done.

To break immediately at the start of a rule:

1. Select **Debug > Break on Rule Start** from the RuleView menu.
   The system displays the breakpoint icon on the line.
2. To continue running the application, select **Debug > Run** from the RuleView menu.

Clear a Breakpoint

To clear a breakpoint in RuleView:

1. With the rule displayed in the rule source window, select the line of code on which to clear a breakpoint.
2. Select **Debug > Toggle Breakpoint** from the RuleView menu, then double-click the selected line.
   The display color changes to identify that the breakpoint has been removed.

or

1. Select *Edit Break/Skip Points* from the **Debug** menu.
2. De-select the break point for the rule to clear.
3. Click **Close** when done.

To continue running the application, select **Debug > Run** or the toolbar or press **F5** .

Clear All Breakpoints

To clear all the breakpoints throughout an application, select **Edit Break/Skip Points** from the **Debug** menu and de-select all the break points for each rule.

Remember that some Restrictions apply to breakpoints.

Restrictions

Because each breakpoint or skippoint is attached to a line number rather than to a particular Rules Language statement, if you change a rule you must verify that the action points are still in the correct positions before debugging.

For example, if you set a breakpoint on line 10 and then add a line of code before line 10, the system attaches the breakpoint to the new line of code. You must manually clear the breakpoint and create a new breakpoint on line 11. Refer to Setting and Clearing a Breakpoint, Setting and Clearing a Skippoint, and RuleView for C Window Settings for procedures on setting and clearing action points.

The following Rules Language statements cannot include action points:

- Comment lines
- Blank lines
- Any lines that include the following statements:
- CASE (but you can set a breakpoint on a CASEOF statement)

- Any type of END (ENDDO, ENDIF, and so forth)
- WHILE
- ELSE
- DCL
- Local variable declarations (between DCL and ENDDCL)

In addition to these, do not use skippoints with the following statements:

- CASEOF – A skippoint placed on the last line of a case causes execution to continue into the next case.
- SQL ASIS
- WHILE loops – A skippoint placed on a line that generates the condition that terminates a WHILE loop results in an infinite loop.

Setting and Clearing a Skippoint

RuleView for C enables skipping a line of code and moving execution to the next line, ignoring rule logic. The way to do this is by defining a *skippoint* . For example, it is possible to skip over a CONVERSE statement to avoid conversing the window, but still execute subsequent statements in the rule.

Some restrictions apply to skippoints. For more information, refer to Restrictions.

Set or Clear Skippoint

To set or to clear a skippoint from RuleView, complete the following steps.

1. Display the rule in the rule source window.
2. Select the line of code on which to set (or clear) a skippoint.
3. Select **Debug > Toggle Skippoint** from the pull-down menu. You can also press F10 or click the **Skip** icon in the toolbar to se a breakpoint. The system displays the skip icon on the line.
4. To set or clear skippoints throughout an application, select **Debug > Edit Break/Skip Points** from the RuleView menu and select or de-select skip points.

Clear All Skippoints

To clear all the breakpoints throughout an application, select **Edit Skippoints** from the **Debug** menu.

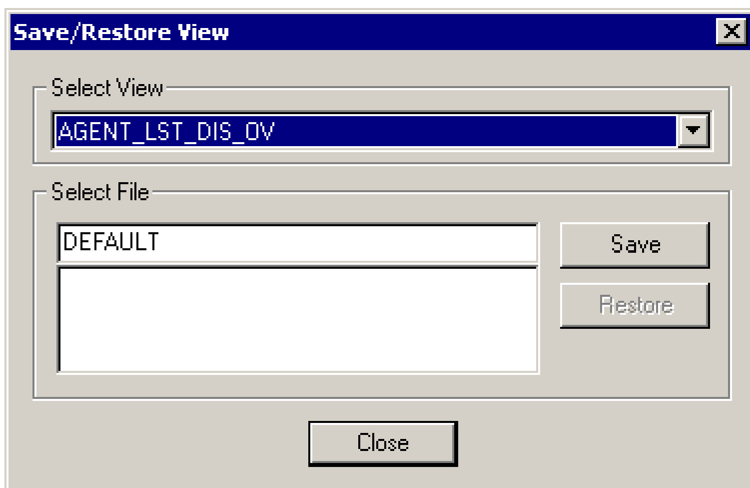
Saving and Restoring Views

You can save a view to a file for later use, such as in another debug session or another rule. Once a view has been saved, it can then be restored for access.

Saving Views

To save a view to a file, select **View** > **Save/Restore Views** . The Save/Restore dialog appears. Select the name of the view to be saved and click **Save** . The filename should be a valid filename without an extension and with a valid subdirectory location.

Figure 4-10 Save/Restore View dialog



Any view in the file running can be saved, but you cannot save single fields. Views are saved with all fields and subviews.

Restoring Views

To restore a view that has already been saved, select **View** > **Save/Restore Views** . The Save/Restore dialog appears. Select one of the files displayed in the list and click the **Restore** button.

RuleView for C Window Settings

You can configure the colors in RuleView for C by editing the **RV.INI** file in the directory where AppBuilder is installed.

Color

The choices of color are listed in the following table:

Table 4-2 Valid colors for RV.INI

Black	DarkBlue	DarkGreen
DarkCyan	DarkRed	DarkMagenta
DarkYellow	DarkGray	Gray
LightGray	Blue	Green
Cyan	Red	Magenta
Yellow	White	

Default

Default properties are listed in the following table:

Table 4-3 Default properties

Setting	Description
RVMaximized	Records whether or not RuleView begins in a maximized state; choices are 0/1 (off/on)
WindowFrame	x(left), y(top), width, height of the window
SaveOnExit	Records whether changes to menu options and window sizes are saved when you exit RuleView; choices are 0/1 (off/on)
ShortName	If on, RuleView displays the rule short name following the long name in the title bar; choices are 0/1 (off/on)

Rule

Rule-source window properties are listed in the following table:

Table 4-4 Rule-source window properties

Setting	Description
WindowFrame	x(left), y(top), width, height of the window
BackColor	Background color of the window
ForeColor	Foreground color of the window
BreakColor	Breakpoint color in rule source window
SkipColor	Skippoint color in the rule source window
CommentColor	Comment color in the rule source window
KeyWordColor	Color for key words in the rule source window
Font	Font used in the window
FontSize	Font size used in the window

View

View window properties are listed in the following table:

Table 4-5 View window properties

Setting	Description
WindowFrame	x(left), y(top), width, height of the window
WinViewOpened	Window View Filter; choices are 0/1 (off/on)
InOutViewOpened	I/O View Filter; choices are 0/1 (off/on)
SysViewOpened	System View Filter; choices are 0/1 (off/on)
AllViewOpened	All Views Filter; choices are 0/1 (off/on)
BackColor	Background color of the window
ForeColor	Foreground color of the window
FieldColor	Color used to display field names in View window
DataColor	Color used to display data in View window
Font	Font used in the window
FontSize	Font size used in the window

Hierarchy

Hierarchy properties are listed in the following table:

Table 4-6 Hierarchy properties

Name	Description
WindowFrame	x(left), y(top), width, height of the window
RuleHierarchy	Records which Hierarchy menu option is selected: (0-4)
BackColor	Background color of the window
ForeColor	Foreground color of the window
Font	Font used in the window
FontSize	Font size used in the window

BreakPoint

This section contains a list of all breakpoints that are currently set. The format is:

```
RuleLongName = 0,line number,1,...
```

This sequence repeats for each breakpoint. Although the line number is the only piece of this sequence actually used by RuleView, the fields before and after the line number must contain placeholder data. As generated, the first number is either a zero (0) or the date the breakpoint was set, and the second is the number one (1).

SkipPoint

This section contains a list of all skippoints that are currently set. The format is:

```
RuleLongName = 0,line number,1,...
```

This sequence repeats for each skippoint. Although the line number is the only piece of this sequence actually used by RuleView, the fields before and after the line number must contain placeholder data. As generated, the first number is either a zero (0) or the date the skippoint was set, and the second is the number one (1).

Exiting RuleView for C

Exiting RuleView for C

To exit the application, select **File > Exit application** from the pull-down menu or press **Alt+F4** . This terminates the application being debugged and exits RuleView.
To exit RuleView without the application that is being debugged terminating, select **File > Exit RuleView** from the pull-down menu or press **Ctrl+F4** . This exits from RuleView.
To exit RuleView the application that is being debugged, click the X in the upper right corner of the RuleView window.

Debugging Batch Applications

Debugging Batch Applications

The batch version of RuleView can be used to debug a mainframe batch application. The batch debugger combines all of the AppBuilder CICS RuleView debugger functions with the added power of ISPF scrolling and unlimited panel display nesting. Invoke the debugger with the TE action from the mainframe.

The batch debugger provides many options for debugging applications. You can set or reset breakpoints to enter the debugger before and after a selected rule, component, or report is called. You can set breakpoints before and after DB2 calls. You can display, change, save, retrieve, and restore views, subviews, and fields. You can also browse the source code of any rule or component.

This topic includes:

- Batch RuleView Commands and Function Keys
- Batch RuleView Panels

Batch RuleView Commands and Function Keys

Batch RuleView Commands and Function Keys

As in other ISPF applications, you control the debugger by entering commands on the command line. Table 5-1 shows the commands in effect when HPSBATCH is selected with the option NEWAPPL(HPSB), as it is in the supplied test method. Type the ISPF *PFSHOW* or *KEYS* command to display the function-key (abbreviated as the letter F with the number of the key) definitions on the panel.

Table 5-1 Batch RuleView default function keys

Key	Command	Description
F3	END	Returns to the previous display or continues if at top level.
F4	RETURN	Exits the debugger and continues processing to the next selected breakpoint.
F6	STOP	Ends debug mode but continues processing. Implies a RETURN.
F7	UP	Scrolls up. Same as ISPF edit and browse.
F8	DOWN	Scrolls down. Same as ISPF edit and browse.
F10	SAVE	Saves the view or field to a test data file on disk.
F11	DATA	Retrieves or deletes from the test data file on disk.
F12	RESTORE	Restores the view or field to its values when first displayed.
F13	INPUT	Displays/modifies the input view of the current rule, component, or report.
F14	OUTPUT	Displays/modifies the output view of the current rule, component, or report.
F15	LOCAL	Displays the local view of the current rule, component, or report.
F17	SKIP	Skips a call to this rule, component, or report. Implies a RETURN.
F18	REPEAT	Repeats a call to this rule, component, or report. Implies a RETURN.
F21	TOGGLE	Expands or contracts the display mode for a view.
F22	SOURCE	Browses the source code of current rule or component.
F23	BREAK	Sets and removes breakpoints for rules, components, or reports.
F24	CURSOR	Positions the cursor at the command line.

	CANCEL	Abends processing immediately.

Batch RuleView Panels

Batch RuleView Panels

The primary debugger displays are:

- Breakpoint Display
- View Display
- Field Display
- Save Display
- Data Display
- Source Code Display

You can use the commands listed in Table 5-1 from any of these panels. However, the data manipulation commands (SAVE, DATA, RESTORE) do not apply to breakpoints, and TOGGLE applies only to views. These panels are stacked in the standard ISPF fashion and displayed again when you issue an END command.
A call trace message is written to the ISPF log file every time you enter the debugger, regardless of whether you set a breakpoint for the particular call. This message is identical to the status line displayed on the debugger panels.

```
POSITION CALL FROM RULENAME TO COMPNAME AT LINE XX
```

- POSITION is either BEFORE, AFTER, SKIPPED, or REPEATING.
- RULENAME is the name of the calling rule or report.
- COMPNAME is the name of the rule, component, or report being called.
- XX is the line number of the USE or CONVERSE statement being executed.

You can display the trace output with the ISPF Dialog Test browse log facility, which is option 7.5 of the primary menu.

Breakpoint Display

The breakpoint selection panel is displayed the first time you enter the debugger, or in response to the BREAK command. A select field, rule, component, or report name and description are displayed on each line. To set a breakpoint, tab to the entity that you want to select and type *Y* over the default *N* next to the name of the entity. To remove a breakpoint, leave the *N* next to the entity. The debugger is activated before and after a selected rule, component, or report is called. Figure 5-1 is the breakpoint display from a simple application.

Figure 5-1 Breakpoint display

```
BREAKPOINTS
    ROW 1 OF 4
 COMMAND
===>                                                            SCROLL ===>
PAGE

    BEFORE CALL FROM HPSBATCH TO RDB2PLI_LONG_NAME AT LINE 0.

    N CDB2PLJ_LONG_NAME                     CDB2PLJ
    N CDB2PLI_LONG_NAME                     CDB2PLI
    N RDB2PLJ_LONG_NAME                     RDB2PLJ
    N RDB2PLI_LONG_NAME                     RDB2PLI
```

View Display

When a breakpoint occurs, the debugger displays the input view if before the call, or the output view if after the call, or the local view if there is neither an input nor an output view. (The breakpoint panel is displayed if there is no corresponding view.) The view panel is also displayed in response to the **INPUT** or **OUTPUT** commands and when a subview is selected from the view panel itself. The contracted mode shows only the next lower level subviews and fields. Expanded mode shows every level. Use the **TOGGLE** command to switch between the two modes. You can change fields by typing over them. Tab down and type an *S* to the left of a subview or field and press Enter to select it. Figure 5-2 shows an output view from the application.

Figure 5-2 Output view display

```
 VIEW VDB2PLI_LONG_NAME                                                  ROW
1 OF 4
 COMMAND ===>                                                          SCROLL
===> PAGE

    BEFORE CALL FROM HPSBATCH TO RDB2PLI_LONG_NAME AT LINE 0.

    01 VDB2PLI_LONG_NAME
    03 MAJOR_PART_LONG_NAME
    03 MINOR_PART_LONG_NAME
    03 QUANTITY_LONG_NAME
```

Field Display

When you select a field from a view display the field panel appears. This is most useful for fields that do not fit on one line. You can change the field data by typing over them. The same data manipulation commands available for views and subviews (SAVE, DATA, and RESTORE) are available for fields. Figure 5-3 shows a display of a field from the view in Figure 5-2.

Figure 5-3 Display of field from output view

```
 FIELD MINOR_PART_LONG_NAME                                          ROW
1 OF 1
 COMMAND
===>                                                        SCROLL ===>
PAGE

    BEFORE CALL FROM HPSBATCH TO RDB2PLI_LONG_NAME AT LINE 0.

    0000
```

Save Display

You can store any view, subview, or field on disk for later retrieval by using the SAVE command from the panel displaying it. Type *SAVE* in the command line and press **Enter** . Supply a 30-character identifier unique to the view or field and a brief description and press **Enter** . This test data is kept in an ISPF table. Figure 5-4 shows a SAVE display for an output view.

Figure 5-4 Save display for an output view

```
 SAVING FIELD MINOR_PART_LONG_NAME                       ENTER INFORMATION
 COMMAND
===>



    BEFORE CALL FROM HPSBATCH TO RDB2PLI_LONG_NAME AT LINE
0.



NAME             ===>




    DESCRIPTION
===>
```

Data Display

Use the DATA command to retrieve or delete stored copies of a view or field. Tab down and type *S* to the left of the identifier, press **Enter** to

select the test data, and press F3 (the END key) to fetch the data. Type *D* to the left of obsolete copies and press **Enter** to delete them. Figure 5-5 shows a test data display containing the view saved in Figure 5-4.

Figure 5-5 Test data display

```
 RETRIEVING FIELD MINOR_PART_LONG_NAME                         ROW 1 OF 1
 COMMAND ===>                                                      SCROLL
===> PAGE

    BEFORE CALL FROM HPSBATCH TO RDB2PLI_LONG_NAME AT LINE
0.

    SELECT                                         A SELECT FROM THE PART
TABLE
```

Source Code Display

The SOURCE command is available from the breakpoint, view, and field displays to examine the source code of the current rule. The ISPF browse system service is used so that any of its subcommands, such as FIND, are active. You can also browse source for any other rule by typing BROWSE RULENAME on the command line. You can display any other data set by specifying BROWSE with no member name. ISPF prompts for the full data set name. Figure 5-6 shows the source code display for our sample rule.

Figure 5-6 Source code display for a sample rule

```
BROWSE -- HPSREP.USER1.RULELIB(RDB2PLI) ------------ LINE 00000000 COL 001 080
********************************* TOP OF DATA  ********************************
USE MODULE CDB2PLI_LONG_NAME
USE MODULE CDB2PLJ_LONG_NAME
USE RULE    RDB2PLJ_LONG_NAME
******************************** BOTTOM OF DATA *******************************
```

Debugging CICS and IMS Applications

This section discusses debugging mainframe applications using RuleView in a mainframe environment. Most of the information in this section applies to the CICS version of RuleView and the IMS version of RuleView.

Use the CICS version of RuleView to debug an AppBuilder application in CICS on the mainframe. Use the IMS version of RuleView to debug an AppBuilder application in IMS on the mainframe.

Because the AppBuilder environment does not support the initiation of CICS RuleView, you must acquire a 3270 session manually invoke CICS RuleView. With Mainframe RuleView, you interactively step through a rules process and examine the contents of views at any point. Use Mainframe RuleView allows to do the following:

- Initiate the execution of a rule
- Break at the interface between a rule and a subordinate rule, window, or component
- Step into the logic of a rule
- Step over the logic of a component, window, or rule
- Step back to the beginning of a component, window, or rule
- Examine and modify the contents of any field within a view
- Review the source code for the active rule
- Save any view data for future reference and reuse

Topics include:

- Mainframe RuleView Panels
- Mainframe RuleView Commands and Function Keys

Mainframe RuleView Panels

In the Mainframe RuleView debugger, there are six different Mainframe RuleView panels. These panels are:

- Rule Selection List: HPR0
- Mainframe RuleView Breakpoint Selection
- Mainframe RuleView Source Code Display
- Mainframe RuleView View Display
- Mainframe RuleView Test Data Maintenance
- Mainframe RuleView Test Data Input

Mainframe RuleView Commands and Function Keys

You can use function keys to jump from one panel to another and to access standard functions. The functions of these keys, described in Mainframe RuleView function keys, can be accessed whenever they are displayed at the bottom of the screen.

Mainframe RuleView function keys

Key	Command	Description
F2	TEST	Displays test data maintenance.
F3	EXIT	Ends the debugger. Continue to test the rule without the debugger.
F4	NAME	Toggles between name and system ID mode.
F5	SAVE	Saves current data permanently as test data.
F6	HEX ON/ HEX OFF	Toggles between character mode and hexadecimal mode.
F7	BKWD	Scrolls backward.
F8	FWD	Scrolls forward.
F9	EXP/CNTRCT	Expands or contracts the display mode for a view.
F10	PROCESS DATA	Processes the test data.
F11	UNDO	Restores all data to what it was when the panel was first displayed.
F12	CANCEL	Goes back to the previous display panel.
F13	IV	Displays or modifies the input view of the current rule, component, or window.
F14	OV	Displays or modifies the output view of current the rule, component, or window.
F15	CONT	Continues processing.
F16	SKIP	Skips processing the current module.
F17	REEX	Re-executes the current rule, window, or component.
F18	LV	Displays or modifies the local view and work view of the current rule.
F19	CREATE DATA	Saves the current view as test data.
F20	ALL	Selects all choices on the display.
F21	GV	Selects a Global View.
F22	SRC	Displays the rule source code display.
F23	BRKPT	Displays the breakpoints selection display.
F24	ABEND	Forces abnormal end of RuleView and rule processing. This stops the execution of the rule and backs out any changes to logged databases.

Rule Selection List HPR0

HPR0, the AppBuilder CICS Rule Selection List facility, displays a list of CICS rules available to be tested. Invoke this facility from a 3270 terminal session with CICS using the HPR0 transaction.

To begin the HPR0 transaction, enter HPR0 from a 3270 terminal. The Rule Selection List panel is displayed, as shown in Rule Selection List panel.

Rule Selection List panel

```
                    R U L E   S E L E C T I O N   L I S T

ENTER SYSTEM ID OR NAME FOR DIRECT SELECTION:
SHORT:               LONG:
ENTER (/)   TO MAKE A SELECTION

        SYSTEM ID         NAME

MORE:    +
      . RAAX1T       HELP_TEXT_TEST_RULE
      . RAAX5B       SET_POPUP_POSITION
      . RAAX5B1      SET_POPUP_POS_NEST_RULE
      . RABEND
      . RARCTES
      . RARC000
      . RARC101
      . RARC102
      . RARC106
      . RASLIST
```

Type the system ID or name of the rule to test or select one of the rules displayed in the list by tabbing to the dot to the left of the rule and entering a slash (/). If you are not sure of the name of the rule, you can type any of the initial characters of the system ID of the rule and press F8 to scroll forward to the rule. Press Enter to process your selection. You can also press F4 to reorder the Rule Selection List by name instead of the implementation name. You can then scroll to the rule name.

Mainframe RuleView Breakpoint Selection

The panel shown in Mainframe RuleView Breakpoint Selection panel appears at the initiation of the first CICS rule of an application. You can also display it at any stage during processing if you need to display or change the Mainframe RuleView breakpoints.

When you select a rule, window, or component as a breakpoint, the application halts immediately before and after execution of that particular rule, window, or component. At these breakpoints, you can examine or modify the data in any view relevant to the application.

The Mainframe RuleView Breakpoint Selection panel lists all the CICS rules, windows, and components to be called during processing. To select a particular breakpoint, tab to the module name and type a slash (/) to the left of the module name. To select all of the modules shown as breakpoints, press `F20` and then `F15` to continue.

To display the panel, press `F23` from any Mainframe RuleView panel that displays the `F23` key.

Mainframe RuleView Breakpoint Selection panel

```
               R U L E V I E W   B R E A K P O I N T   S E L E C T I O N

 POSITION IN RULE: ACCT_MAINT_LIST_BUILD   BEFORE EXECUTION OF
 ENTITY:   ACCT_MAINT_LIST_BUILD

 SELECT (/) ONE OR MORE ENTITIES.   THEN PRESS CONTINUE


            MORE:    +
RULE       ACCT_MAINT_LIST_BUILD                    RAML005
RULE       ACCT_MAINT_LIST_DSPLY                    RADL001
RULE       ACCT_MAINT_SORT_SLCT_DRV                RAMSSD1
RULE       ACCT_MAINT_START_BUILD                   RAMS001
RULE       ACCT_MAINT_WRKLD_ID_READ_SQL          RMAM014
RULE       ACCT_MONTR                                    RAM003
RULE       ACCT_MONTR_LIST_BUILD                    RAMLB01
RULE       ACCT_MONTR_LIST_DSPLY                    RAMD001
RULE       ACCT_MONTR_SORT_SLCT_DRV                RASSD2
RULE       ACCT_MONTR_START_BUILD                   RAMSB0
RULE       ACCT_NBR_EOM_PART_READ_SQL             RANE001
```

Mainframe RuleView Source Code Display

To display this panel, press F22 from any Mainframe RuleView panel that displays the F22 key. A complete listing of the original rule source code of the currently executing rule is displayed, and the current position in the rule logic is highlighted (Mainframe RuleView Source Code Display panel).

Mainframe RuleView Source Code Display panel

```
                 R U L E V I E W   S O U R C E   C O D E   D I S P L A
 Y

  SOURCE LISTING FOR RULE:   ACCT_MAINT_LIST_BUILD      MORE:    +
  BEFORE       EXECUTING   RULE                      : ACCT_MAINT_LIST_BUILD
  *************************************************************
 *> .$.0 040490 Edited Clone of RPL022 RXL022A2.SRC 1 NS    <*
 *>Fix Subscript: VPL012(I - CURRENT_BUFFER + 1) with WtW JAK <*
    *>
              <*
  dcl

       CURRENT_BUFFER ,I,J,k, WtW   smallint;   *> WtW temp subscript   JAK
 <*
       MODE char(1);
       FLAG smallint;
       P integer;
  enddcl

 map 'VPL002X' to VIEW_LONG_NAME OF GET_ELEVATOR_POSITION_I
 map 'VPL002X' to VIEW_LONG_NAME OF SET_VIRTUAL_LISTBOX_SIZE_I
 map 'VPL002X' to VIEW_LONG_NAME OF SET_LAST_VISIBLE_OCCURRENCE_I
```

Mainframe RuleView View Display

To display this panel (Mainframe RuleView View Display panel) at any stage during Mainframe RuleView processing, press *F13* for the input view, *F14* for the output view, or *F18* for the local view. You see the input or output of a rule, window, or component in the form defined by the view and field structure in the repository. You can also overwrite the contents of any field in the view. The input is checked to prevent entering any data incompatible with the defined field type. The local view for a rule can only be displayed at a breakpoint within the rule. This topic includes:

- Compacted and Expanded Modes
- Other View Display Commands

Compacted and Expanded Modes

There are two possible view display modes: compacted and expanded. Compacted mode is the default?only level 01 and level 03 data items are displayed. Press *F9* to see the expanded mode, which displays all particular view levels. To expand a specific occurrence, place the cursor on that occurrence and press *Enter* .

Expanded mode displays all the fields contained in a view together with all the fields contained in any subviews. This mode is generally more convenient for viewing a complex series of small subviews.

The maximum amount of data that can be displayed on the standard output screen is 50 bytes. If a field contains data longer than this, the > symbol is displayed next to the field. To display or update the expanded contents of the field, move the cursor to a position level with the field definition and press *Enter* . (Data that a 3270 device cannot display appears as a period.)

Mainframe RuleView View Display panel

```
                  R U L E V I E W   D I S P L A Y

 POSITION IN RULE:    SQL_INS_HPS_OBJ_CLASS
 ENTITY:  SQL_INS_HPS_OBJ_CLASS          WILL BE EXECUTED
NEXT
 VIEW:  SQL_INS_HPS_OBJ_CLASS_I
 LVL FIELD/VIEW NAME        CONTENTS

        MORE:
 01 SQL_INS_HPS_OBJ_CLASS_
 03 HPS_STND_OBJ_CLASS_D
 05 T_HPS_OBJ_TYPE
 05 N_HPS_OBJ_CLASS_NME
 05 T_HPS_OBJ_CLASS_ABBR
 05 C_HPS_OBJ_CLASS_RANK    +00000
 < 05
T_HPS_OBJ_CLASS_DESC
        >
 05 I_CREATE_BY
 05 D_CREATE_DTE            00/00/00
 05 I_MAINT_BY
 05 D_MAINT_DTE             00/00/00
```
**

Other View Display Commands

To refresh the original contents of the view, press *F11* (Undo). This overwrites the view contents with the values that were current when the breakpoint was first reached (that is, before any changes).

To toggle between the input and output views to compare them at any given breakpoint, press *F13* and *F14* .

To change view display modes from compacted to expanded or vice versa, press *F9* . The display in both cases is repositioned at the beginning of the view definition. The view display is initially in compacted mode.

Mainframe RuleView Test Data Maintenance

The Mainframe RuleView Test Data Maintenance panel (Mainframe RuleView Test Data Maintenance panel) allows you to save the current contents of a view as test data, retrieve previously saved data for the current view, or delete previously stored test data.

To display only the panel, press *F2* from the Mainframe RuleView Display panel (Mainframe RuleView View Display panel).

Mainframe RuleView Test Data Maintenance panel

```
      R U L E V I E W   T E S T   D A T A   M A I N T E N A N C E

 CURRENT POSITION BEFORE WPL012
 VIEW: RENT12                           RULE: CALC_RENTAL_FEE

 TO PROCESS TEST DATA FOR CURRENT VIEW ENTER (/) TO RETRIEVE OR (-) TO
DELETE
                                                             MORE:
 _ DATA0325 Test Data from March 24
 _ DATA0326 Test Data from March 26
 _ TESTGAIL Gails test data
 _ TEST9877 This is test data
 _ TRDC3002 SECOND TEST OF CALC_RENTAL_FEE
 _ TRDC3007 TEST SEVEN FOR CALC_RENTAL_FEE
```

Saving and Retrieving Data

Press *F19* from the Test Data Maintenance panel to save the current contents of the view permanently in a VSAM file for reuse as test data. The Test Data Input panel is displayed, prompting for a name and description of the test data to be saved. Press *F5* to save this data.
The Test Data Maintenance panel displays a scrollable list of names of previously saved test data. To replace the contents of the current view with the desired test data, type a forward slash (/) next to the desired test data entry.

- To process the selected data, press *F10.*
- To delete the selected test data from the Test Data VSAM file, type a hyphen (-) next to the test data entry and press *F10.*
- To create a new instance of test data, press *F19.*
- To exit the Test Data Maintenance display, press *F12.*

Mainframe RuleView Test Data Input

The Mainframe RuleView Test Data Input panel (Mainframe RuleView Test Data Input panel) prompts you for a name and a description for the test data to be saved. The name is used in the Test Data Maintenance panel (Mainframe RuleView Test Data Maintenance panel) to retrieve or delete the test data.

After you type the name and description of the test data, press *F5* to save it. Press *F12* to cancel the panel without saving the test data.

Mainframe RuleView Test Data Input panel

```
             R U L E V I E W   T E S T   D A T A   I N P U T


 ENTER NAME AND DESCRIPTION OF TEST DATA FOR
 VIEW: RENT12                           RULE: CALC_RENTAL_FEE

        NAME:

 DESCRIPTION:
```