Using Apama Studio

5.2.0

August 2014

This document applies to Apama 5.2.0 and to all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Document ID: PAM-Using_Apama_Studio-5.2.0-20140808@233876

Table of Contents

Preface

About this documentation

Apama Studio is an integrated environment for developing Apama applications. The process of developing an Apama application is centered around an Apama *project*. This manual describes how to create projects and add the various Apama resources that make up an application.

Preface

How this book is organized

The information in this book is organized as follows:

- "Overview of Developing Apama Applications" on page 13 describes Apama Studio's integrated environment for developing Apama applications.
- "Working with Projects" on page 29 describes how to create and configure projects, add resources to them, and launch applications from them.
- "Creating Blocks" on page 81 describes how to create custom blocks for scenarios in Apama applications.
- "Launching Projects" on page 102 describes how to create launch configurations projects.
- "Debugging EPL Applications" on page 125 describes how to use Apama Studio to debug applications written in the Apama Event Processing Language (EPL). Beginning with Release 4.3, Event Processing Language is the new name for MonitorScript.
- "Debugging JMon Applications" on page 134 describes how to use Apama Studio to debug Apama applications written in JMon.
- "Profiling EPL Applications" on page 146 describes how to use Apama Studio to collect profiling data for applications written in EPL.
- "Using the Data Player" on page 158 describes how to configure the Apama Database Connector (ADBC) adapters to play back event data from a database and how to use the Query Editor and Data Player control.
- "Generating Dashboards" on page 168 describes how to use the Dashboard Wizard to generate dashboards and how to specify the dashboard's visualization objects and the scenario variables to use with them.

- "Generating Dashboards" on page 168 describes how to prepare a project's dashboards for deployment, including how to create a deployment package with the Dashboard Deployment Configuration Editor.
- "File Definition Formats" on page 188 describes the formats of Apama's function definition and block definition files.

Preface

Documentation roadmap

On Windows platforms, the specific set of documentation provided with Apama depends on whether you choose the Developer, Server, or User installation option. On UNIX platforms, only the Server option is available.

Apama provides documentation in three formats:

- HTML viewable in a Web browser
- PDF
- Eclipse Help (if you select the Apama Developer installation option)

On Windows, to access the documentation, select Start > All Programs > Software AG > Apama 5.2 > Apama Documentation . On UNIX, display the `index.html` file, which is in the `doc` directory of your Apama installation directory.

The following table describes the PDF documents that are available when you install the Apama Developer option. A subset of these documents is provided with the Server and User options.

Title	Contents
What's New in Apama	Describes new features and changes since the previous release.
Installing Apama	Instructions for installing the Developer, Server, or User Apama installation options.
Introduction to Apama	Introduction to developing Apama applications, discussions of Apama architecture and concepts, and pointers to sources of information outside the documentation set.
Using Apama Studio	Instructions for using Apama Studio to create and test Apama projects; write, profile, and debug EPL programs; write JMon programs; develop custom blocks; and store, retrieve and playback data.
Developing Apama Applications in Event Modeler	Instructions for using Apama Studio's Event Modeler editor to develop scenarios. Includes information about using standard functions, standard blocks, and blocks generated from scenarios.
Developing Apama Applications in EPL	Introduces Apama's Event Processing Language (EPL) and provides user guide type information for how to write EPL programs. EPL is the native interface to the correlator. This

Title	Contents
	document also provides information for using the standard correlator plug-ins.
Apama EPL Reference	Reference information for EPL: lexical elements, syntax, types, variables, event definitions, expressions, statements.
Developing Apama Applications in Java	Introduces the Apama in-process API for Java, referred to as JMon, and provides user guide type information for how to write Java programs that run on the correlator. Reference information in Javadoc format is also available.
Building Dashboards	Describes how to create dashboards, which are the end-user interfaces to running scenario instances and data view items.
Dashboard Property Reference	Reference information on the properties of the visualization objects that you can include in your dashboards.
Dashboard Function Reference	Reference information on dashboard functions, which allow you to operate on correlator data before you attach it to visualization objects.
Developing Adapters	Describes how to create adapters, which are components that translate events from non-Apama format to Apama format.
Developing Clients	Describes how to develop C, C++, Java, or .NET clients that can communicate with and interact with the correlator.
Writing Correlator Plug-ins	Describes how to develop formatted libraries of C, C++ or Java functions that can be called from EPL.
Deploying and Managing Apama Applications	Describes how to: • Use the Management & Monitoring console to configure, start, stop, and monitor the correlator and adapters across multiple hosts. • Deploy dashboards over wide area networks, including the internet, and provide dashboards with effective authorization and authentication. • Improve Apama application performance by using multiple correlators, and saving and reusing a snapshot of a correlator's state. • Use the Apama ADBC adapter to store and retrieve data in JDBC, ODBC, and Apama Sim databases. • Use the Apama Web Services Client adapter to invoke Web Services. • Use correlator-integrated messaging for JMS to reliably send and receive JMS messages in Apama applications.

Title	Contents
	• Use Universal Messaging to connect correlators.
Using the Dashboard Viewer	Describes how to view and interact with dashboards that are receiving run-time data from the correlator.

Preface

Contacting customer support

You may open Apama Support Incidents online via the eService section of Empower at http://empower.softwareag.com. If you are new to Empower, send an email to `empower@softwareag.com` with your name, company, and company email address to request an account.

If you have any questions, you can find a local or toll-free number for your country in our Global Support Directory at https://empower.softwareag.com/public_directory.asp and give us a call.

Preface

Chapter 1: Overview of Developing Apama Applications

Apama Studio is an integrated environment for developing Apama applications. The process of developing an Apama application is centered around an Apama *project*. In Apama Studio you create a project; you then use Apama Studio to:

- Create new Apama resources for the application
- Include standard, pre-packaged Apama resources
- Include existing Apama resources from other projects or applications
- Specify configuration properties necessary for launching the application
- Run and monitor the application
- Export the initialization information necessary for deploying the application

When you add resources to your application, Apama Studio creates the resource's metadata and launches the appropriate editor where you add the code to implement its behavior. As you create the resource's application code, Apama Studio automatically validates it. Where necessary, Apama Studio launches the Apama tool such as Event Modeler or Dashboard Builder that is appropriate to the specific resource being added.

Samples and tutorials

Apama Studio is packaged with several sample programs and tutorials. These are available from the Apama Welcome screen, which is displayed when you start Apama Studio for the first time and which is always available by selecting Help > Welcome from the Apama Studio menu. From the Welcome screen click Samples or Tutorials and then click Apama Samples or Apama Tutorials.

The Apama samples are demonstration applications that illustrate some of the features and capabilities of the Apama platform; they include dashboards in which you can interact with the application and readme files that describe what the application does, what files make up the project,

and how you might modify the application. Use the demonstration applications to gain an overview of what goes into an Apama project and what some common Apama applications can do.

The tutorials are interactive instructions that get you quickly up to speed writing Event Processing Language programs, creating scenarios, defining reusable blocks for use in scenarios, and creating dashboards that provide the user interface to scenarios. Each tutorial provides a skeleton project and a completed project. At the end of the tutorial instructions, you run the project.

When you open a sample or a tutorial for the first time, it is *copied* into the Eclipse workspace. You can revert to the original sample or tutorial without any changes you've made at any time by deleting the project as follows:

1. Right-click the project.
2. Click Delete.
3. Select Also delete contents in the confirmation dialog and click Yes.

Then open the project again from the Tutorials or Samples Welcome page.

Overview of Developing Apama Applications

The Apama interface

Apama Studio is an Eclipse plug-in and using it is similar to working in other Eclipse development perspectives.

Apama Studio provides the following distinct perspectives for working with projects, **Apama Developer**, **Apama Runtime**, **Apama Workbench**, and **Apama Profiler**. In each of the perspectives, you can create projects, add Apama resources, and launch your application. While developing your application you can switch from one perspective to the other. For more information, see the following:

- "The Apama Workbench perspective" on page 15
- "The Apama Developer perspective" on page 17
- "The Apama Runtime perspective" on page 18

When you debug an Apama application, by default Apama Studio switches to the Eclipse **Debug** perspective. When profiling an Apama application written in the Apama Event Processing Language (EPL), Apama Studio uses the **Apama Profiler** perspective. (Beginning with Apama 4.3, "Apama Event Processing Language" is the new name for MonitorScript.)

Note: When using any of the Apama perspectives, you can redisplay the default perspective layout by selecting Window > Reset Perspective from the Apama Studio menu.

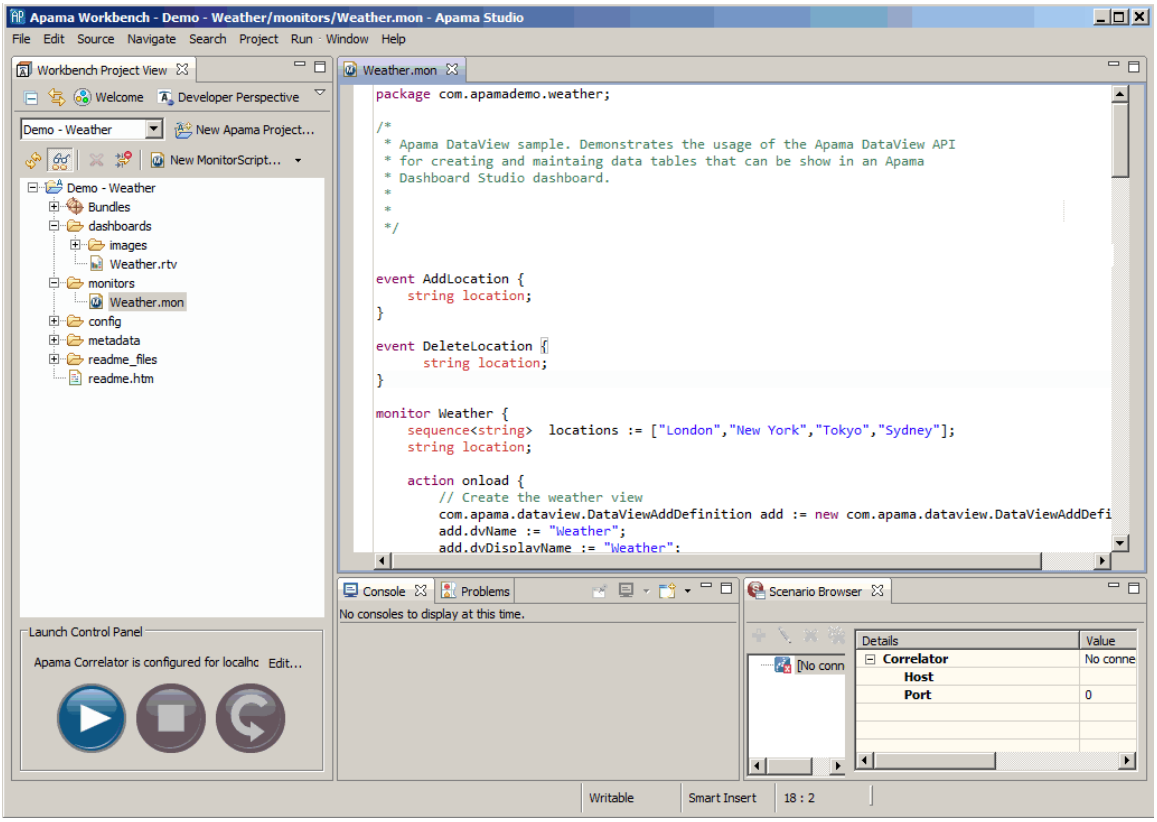
Caution: The recommended installation folder is `Program Files`, which is a protected location on recent Microsoft Windows operating systems. These include the client operating systems Windows 7 and Windows 8.1, and the server operating systems Windows Server 2008 R2 and Windows 2012 R2. To write to the `Program Files` folder, you must run the Apama installer with Administrative privileges. After Apama installation, if you want to add additional plug-ins to Eclipse, you can run the Eclipse plug-in installer or use the Eclipse Check for Updates facility but you must have Administrative privileges when you install the Eclipse plug-in. Lack of Administrative privileges might cause the plug-in installation to fail or become corrupt. Adiministrative privileges are

required because Eclipse also installs its plug-ins in the protected `Program Files`s folder. Alternatively, you can choose to install Apama in a non-recommended location outside the `Program Files` folder.

Overview of Developing Apama Applications

The Apama Workbench perspective

The **Workbench** perspective presents a more streamlined view of a single Apama project. It shows only the resources related to that project and provides a simplified way of launching an Apama application.



The important interface components of the **Apama Workbench** perspective are:

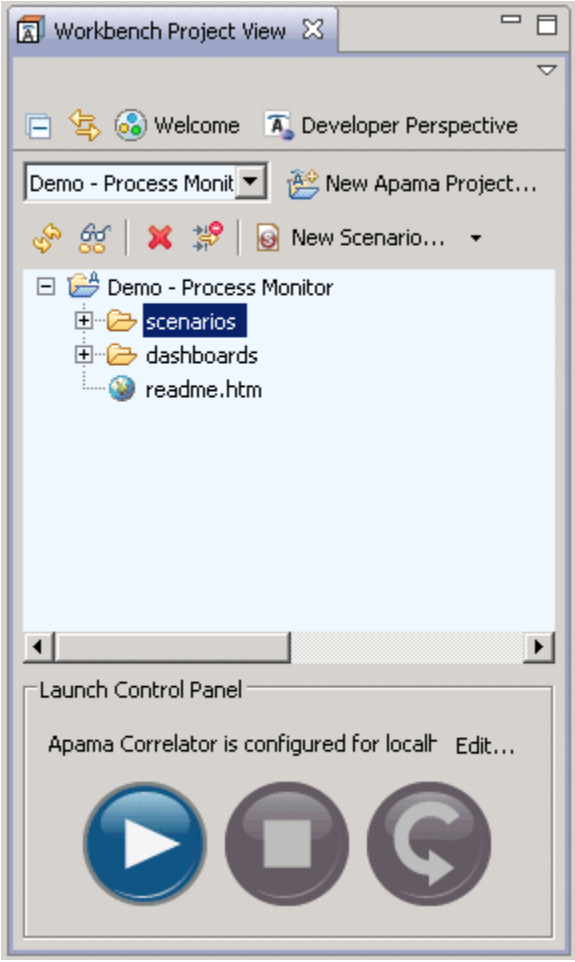
- "Workbench Project view" on page 15
- "Editors" on page 21
- "Scenario Browser view" on page 22
- "Console view" on page 25
- "Problems view" on page 26

Overview of Developing Apama Applications

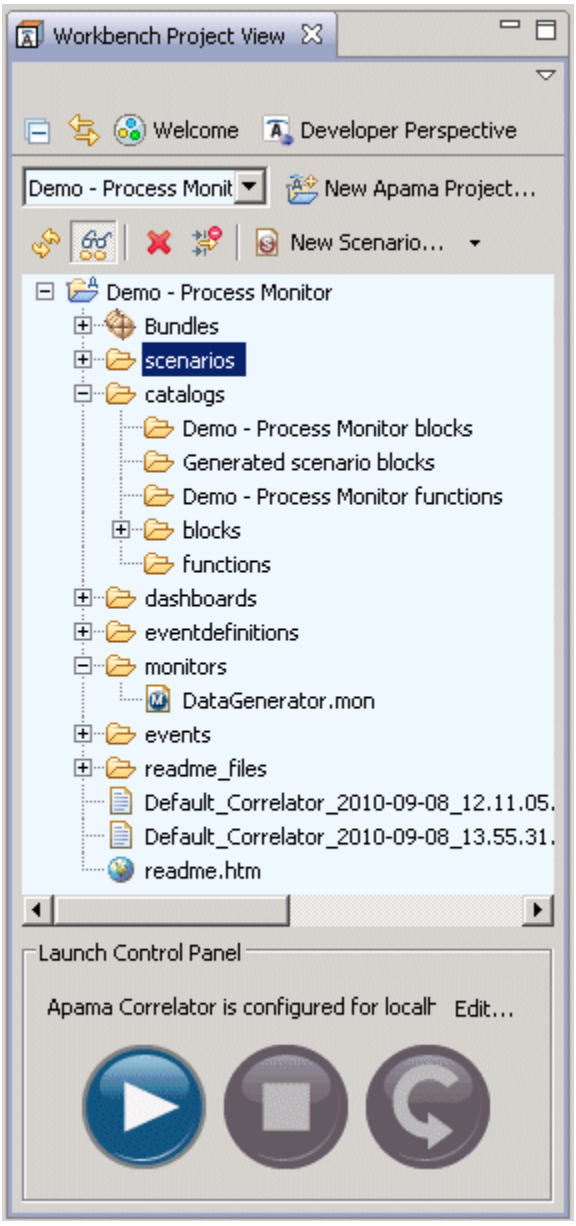
Workbench Project view

When you develop an application in the **Apama Workbench** perspective, you work on a single project at a time. The project is displayed in the **Workbench Project** view. This is where you add the various Apama resources that are necessary for the application. This is also where you run the application that is represented by the project.

When viewing a project in the **Workbench** perspective you can use the Show All Folders icon to toggle between a displaying a limited view of the project's resources and displaying all of the project's resources. In the limited view, below, Apama Studio only displays scenarios and dashboard resources:



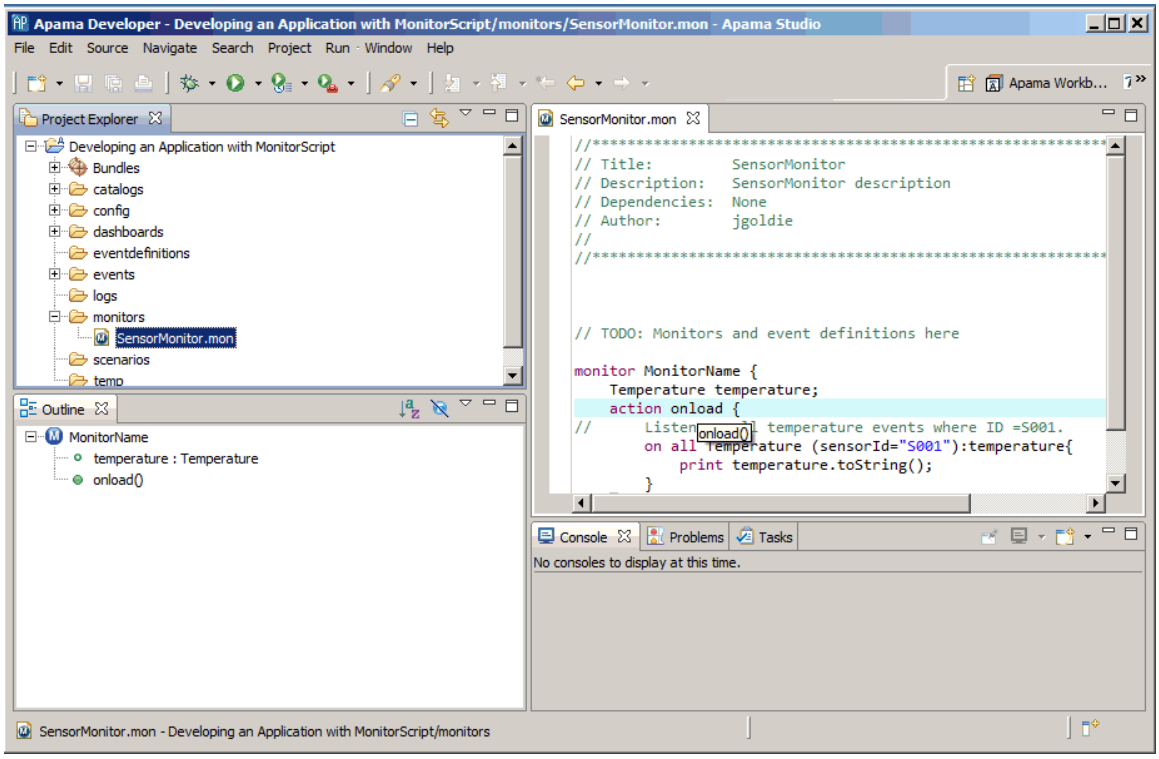
Clicking the Show All Folders icon displays an expanded view that lists the Event Processing Language (EPL) files and other file types, such as event files, block files, and function files used in the project as shown below.



The Apama Workbench perspective

The Apama Developer perspective

The **Apama Developer** perspective uses the full Eclipse interface and displays all the Apama projects in the user's workspace. It is designed for experienced developers and assumes that you are familiar with standard Eclipse features.



The important interface components of the Developer perspective are:

- "Project Explorer view" on page 18
- "Console view" on page 25
- "Outline view" on page 22
- "Problems view" on page 26
- "Editors" on page 21

Overview of Developing Apama Applications

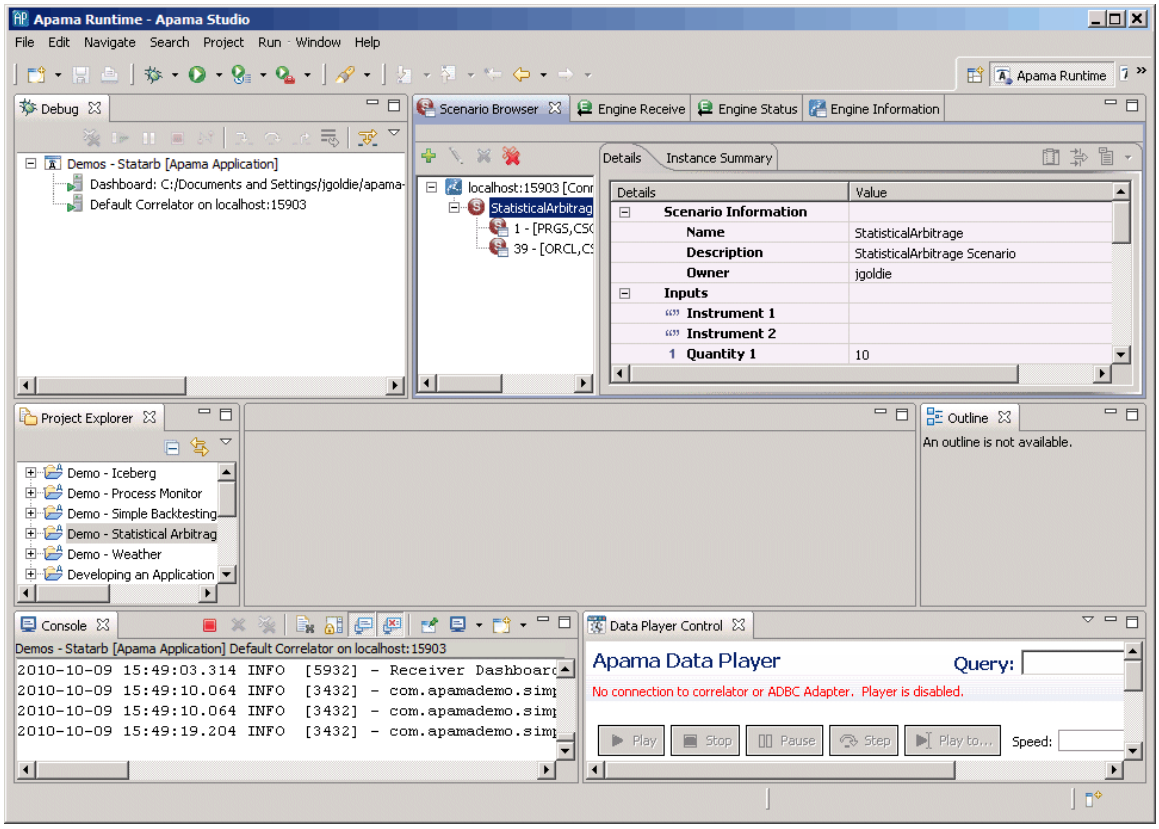
Project Explorer view

The **Apama Developer** perspective uses the standard Eclipse **Project Explorer** view. This view is at the center of the process of developing an application in Apama Studio. This is where you add and manage all the Apama resources that are necessary for the application. The **Project Explorer** view is where you build and launch your projects.

The Apama Developer perspective

The Apama Runtime perspective

The **Apama Runtime** perspective is similar to the **Apama Developer** perspective but is designed for inspecting and interacting with a running Apama application. This perspective is designed for experienced developers and assumes that you are familiar with standard Eclipse features.



To use the **Apama Runtime** perspective:

1. Select Window > Open Perspective > Other from the Apama Studio menu.
2. Select Apama Runtime from the **Open Perspective** dialog.
3. Click OK.

The Apama Runtime perspective is made up of these views:

- "Project Explorer view" on page 18
- "Scenario Browser view" on page 22
- "Engine Receive view" on page 23
- "Engine Status view" on page 24
- "Engine Information view" on page 25
- "Outline view" on page 22
- "Console view" on page 25
- "Problems view" on page 26
- "Data Player Control view" on page 26

Overview of Developing Apama Applications

Apama projects

An Apama project typically manages a single Apama application. A project provides a means of keeping the application's resources organized. In the process of developing an application with Apama Studio, you add the various resources that make up the application to the project. For example, you can include:

- EPL files — These files define *monitors* and associated *event types* that are used by your application. EPL files have a `.mon` extension.
- JMon Monitor files — These Java source files define *monitors* for Apama applications written in Java.
- JMon Event files — These Java source files define *event types* for Apama applications written in Java.
- Scenario definition files — If your application uses scenarios, you specify the new scenario in Apama Studio. This adds a *scenario definition file* to your project and opens the new scenario in Apama Studio's Event Modeler editor where you specify its behavior. Scenario definition files have a `.sdf` extension.
- Dashboard definition files — If your application uses dashboards, you specify the new dashboard in Apama Studio. This adds a *dashboard definition file* to your project and opens the new dashboard in Apama's Dashboard Builder. Dashboard definition files have an `.rtv` extension.
- Bundles — These are pre-packaged collections of Apama objects.
- Event files — Event files have an `.evt` extension.
- Block definition files — If your application uses scenarios and, in addition to the standard blocks packaged with Apama, you need to create customized blocks, you create the block in Apama Studio. This adds a *block definition file* to your project. Block definition files have a `.bdf` extension.
- Function definition files — If your application uses scenarios and, in addition to the standard functions packaged with Apama, you need to create customized functions, you create the function in Apama Studio. This adds a *function definition file* to your project. Function definition files have a `.fdf` extension

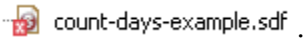
Overview of Developing Apama Applications

Managing project hierarchies

Apama Studio organizes your project into a hierarchy that is displayed in the **Project Explorer** view (if you are using the Developer perspective) or the **Workbench Project** view (if you are using the Workbench perspective). The hierarchy is made up of folders that group Apama resource files by type. The **Project Explorer** view lists all your Apama projects, while the Workbench Project view lists just the current project. Apama Studio displays a different icon for each type of resource file:

-  indicates an Apama project.
-  indicates a bundle.
-  indicates an EPL (`.mon`) file.
-  indicates a JMon monitor or JMon event (`.java`) file.

- indicates an EPL plug-in written in Java (`.java`) file.
- indicates an event (`.evt`) file.
- indicates a scenario (`.sdf`) file.
- indicates a scenario block (`.bdf`) file.
- indicates a scenario function (`.fdf`) file
- indicates a dashboard (`.rtv`) file.
- indicates a correlator deployment package (`.cdp`) file.

If a monitor or scenario file contains an error, Apama Studio displays an error icon over the monitor file name or scenario file name, over the folder that contains the file that has the error, and over the project folder. For example, a scenario file that contains an error would look like this:


Apama projects

Working with Apama projects

Apama Studio provides a variety of ways to interact with the resources in your application's project.

This section briefly describes these Apama Studio features; for more information on how to use them, see "Working with Projects" on page 29 and "Launching Projects" on page 102.

Overview of Developing Apama Applications

Editors

Apama Studio provides wizards and editors to create and modify the resource files that define the following:

- Monitors
- Events
- Blocks
- Functions
- Scenarios
- Dashboards

When you add a resource, Apama Studio generates the definition file and opens it in the appropriate Apama editor.

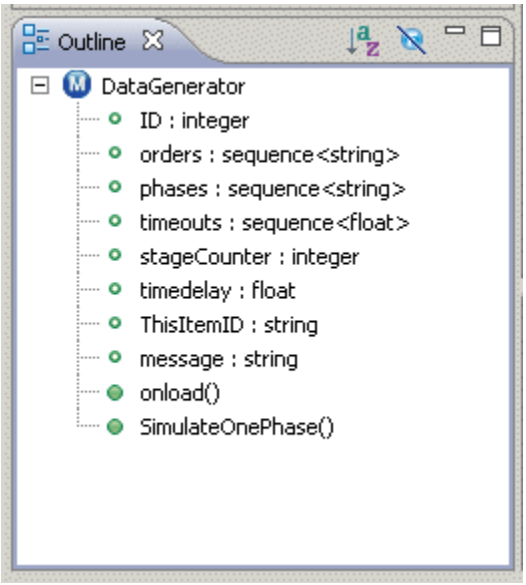
Note: When you add or edit a dashboard, Apama Studio creates the dashboard's definition file and opens the Apama Dashboard Builder in a separate window.

As you edit resource files, you can take advantage of the Apama Studio editing features, including content assistance, auto-bracketing, templates for frequently entered constructs, and problem detection. After you build an Apama project, Apama Studio flags each line that contains an error.

Working with Apama projects

Outline view

The **Outline** view shows the structure of the EPL or block file that is open in an Apama editor. The following illustration displays an **Outline** view of an EPL file.

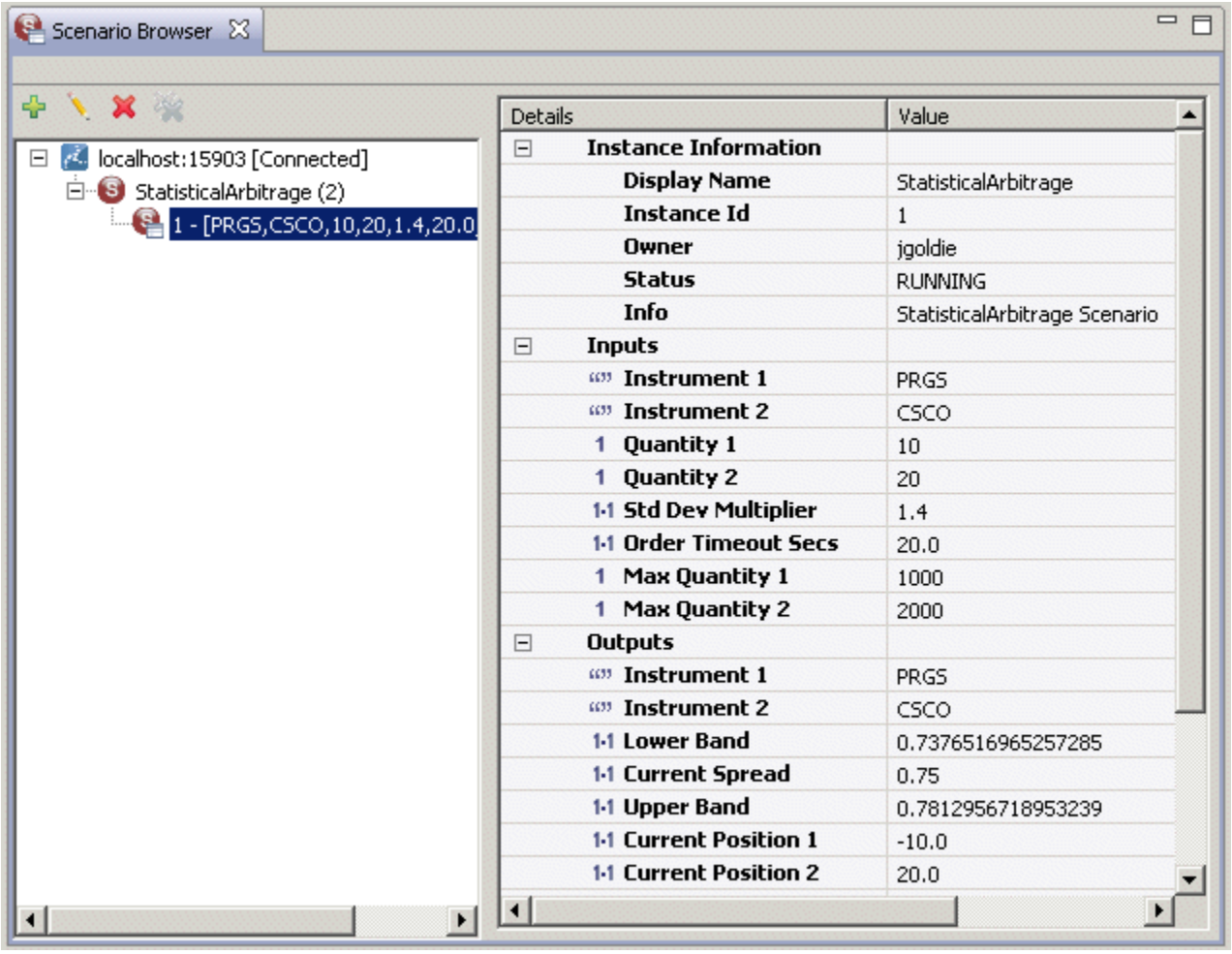


When you click on an event type or action in the outline, the editor pane highlights and displays the line of code that defines that item in the file.

Working with Apama projects

Scenario Browser view

The **Scenario Browser** view displays scenario definitions that are loaded in the correlator and any scenario instances that are running. When an application is running in Apama Studio, you can add, edit, and delete scenario instances.



For more information on using the view, see "Using the Scenario Browser view" on page 118.

Working with Apama projects

Engine Receive view

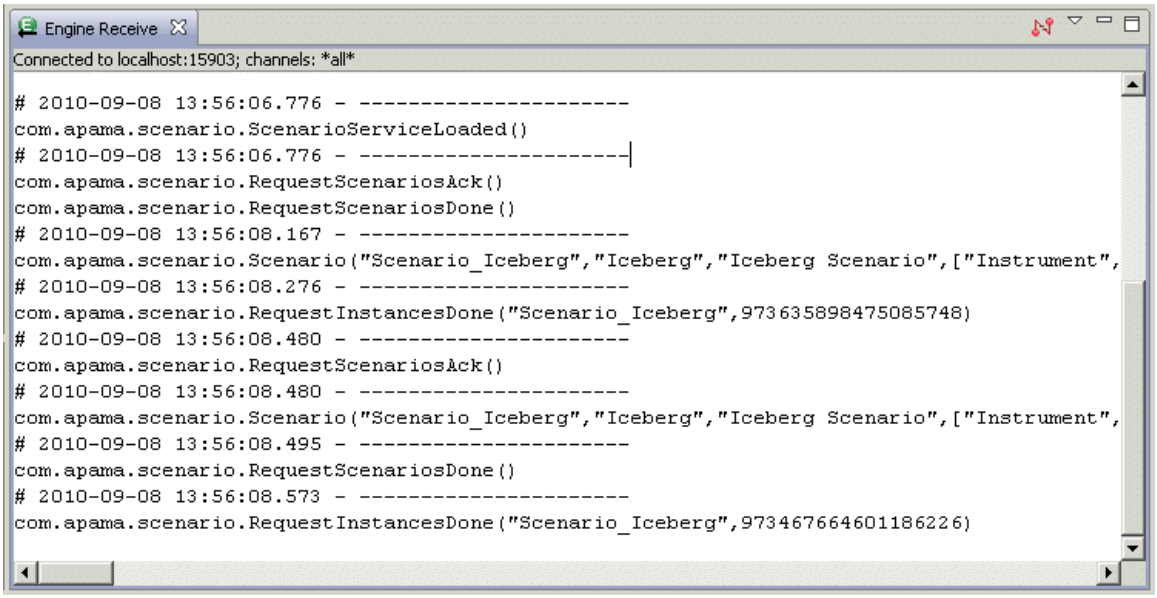
The Engine Receive view shows all events generated from the connected correlator.

The Toggle Connection button  to temporarily disconnect the EngineReceive view from the correlator.

When the Toggle Connection button is pressed, the console will not update any further events from correlator. The background of the console will be changed to indicate the temporary disconnect mode. The Select All, Clear, and Copy actions will still work in this mode.

To resume from the temporary disconnect mode, simply click on the Toggle Connection button again to resume the connection. The console will be cleared and new events will be shown in the console again.

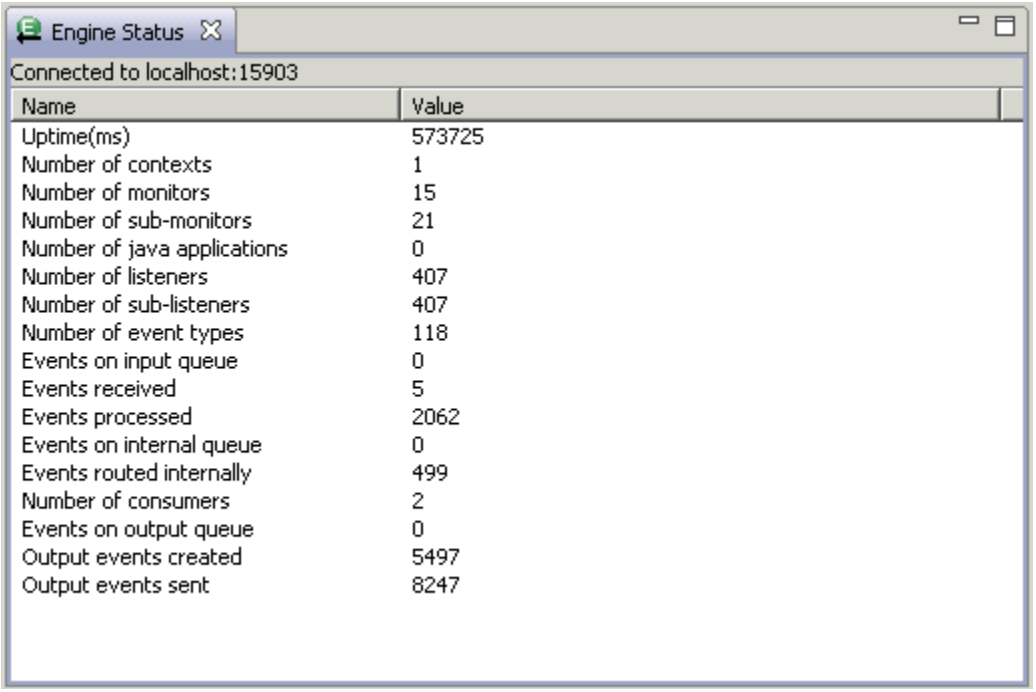
For more information on the Engine Receive view, see "Using the Engine Receive view" on page 116.



Working with Apama projects

Engine Status view

The Engine Status view displays the information about the correlator status. The information is the same as the output of Apama command line tool `engine_watch`.

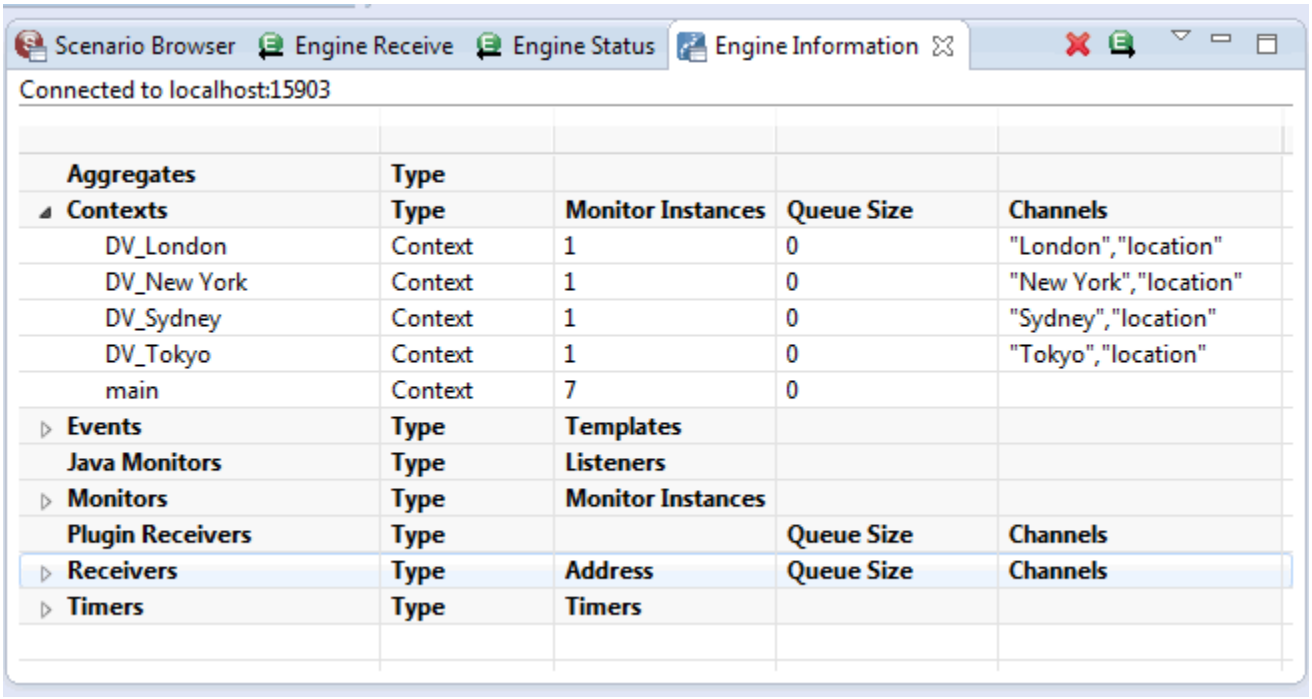


For more information on Studio's Engine Status view, see "Using the Engine Status view" on page 117.

Working with Apama projects

Engine Information view

The **Engine Information** view inspects a running correlator and displays defined contents of the correlator. It shows the same information as the Apama command line tool `engine_inspect`. For example:



The following actions are available:

- Delete button — Direct deletion of the defined named entities. You can also delete the entity by right clicking the entity and selecting Delete from the drop down menu. A confirmation message is displayed after you click Delete. If the entity has dependencies on it, a confirmation message is displayed asking if you want allow a forced deletion.
- Send button — Send an event from this view. You can also send an event by right clicking it and selecting Send from the drop down menu.

For more information on Studio's **Engine Information** view, see "Using the Engine Information view" on page 115.

Working with Apama projects

Console view

The **Console** view displays information concerning a running Apama application. An application can have several consoles:

- Correlator — Displays output from the correlator.

- Engine Inject —Displays initialization information injected to the correlator.
- Engine Send — Displays information from Apama components such as dashboards that stream data to the correlator.
- Correlator Initialization — Displays information about the correlator initialization including the Java files, `.mon` files (monitors), and `.sdf` files (scenarios) that have been injected and the `.evt` files (events) that have been sent and whether the actions succeeded or failed.

To view one of these consoles, click the drop down arrow of the Display Selected Console button and select the console you want.

Working with Apama projects

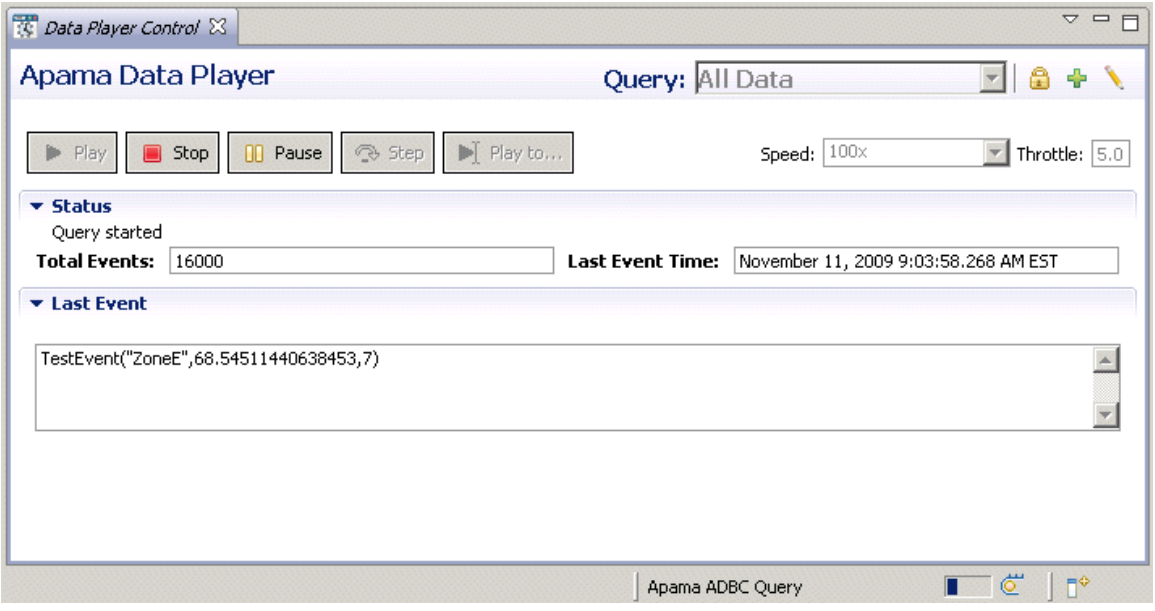
Problems view

The **Problems** view is a standard Eclipse **Problems** view that lets you view a list of any errors in your EPL and scenario files, and ODBC-ADBC and JDBC-ABDC adapter instances. When you build an Apama project, if Apama Studio detects errors in your project's files, or file dependency errors, it lists them here. The **Problems** view is displayed when you run an application.

Working with Apama projects

Data Player Control view

The **Data Player Control** view allows you to control how Apama data is played back in an application running in a project. For example, you can specify how fast to play back the event data, step through events one at a time of specify a query to filter what events are played back.



For more information on the Data Player Control, see "Using the Data Player" on page 158.

Working with Apama projects

Building Apama projects

Apama Studio validates Apama projects, in a process known as “building” the project. During validation, Apama Studio

- Checks the syntax to ensure that it is valid EPL or Java code.
- Checks that only valid values have been specified. For example, if you specify integer as the type of a field, it ensures that you specify an integer as the value of the field.
- Ensures that the dependencies are valid throughout the project.

In summary, Apama Studio validates that its launcher can inject the project’s EPL files into the correlator.

For more information, see "Building Apama projects" on page 64.

Overview of Developing Apama Applications

Launching Apama projects

Apama Studio can launch an Apama project in a test environment. When Apama Studio launches a project, the default is that it starts an instance of the correlator and injects all the resources that are included in the project. You can change the default launch behavior by editing the launch configuration for the project.

You can specify multiple launch configurations for each Apama project, including configurations for debugging and profiling applications. For each launch configuration, you can:

- Add correlator arguments.
- Specify event files that should be sent to initialize/start the application.
- Specify what adapters to use.
- Define environment variables, for example, appending directories to the `PATH` used to start the correlator and any IAF processes.
- Indicate whether the configuration can be shared among Apama Studio installations.

For more information on creating launch configurations and launching Apama projects, see "Launching Projects" on page 102.

Overview of Developing Apama Applications

Specifying the location of the license file

When Apama Studio launches an application configured to start a correlator Apama Studio looks for the Apama license file in the default location, which is `APAMA_WORK\license\license.txt`. This is automatically taken care of when the correlator is started by Apama Studio.

If your Apama license file is not located in the default location you can specify the location before launching your application. If you do not then the correlator will be started in an evaluation mode. This means it can communicate only with processes on the same machine and it will automatically exit after 30 minutes of operation. Check the correlator console output on startup to see if the correlator is running in evaluation mode. If it is put a license into the default location or provide the location of the license file if it is somewhere non-standard.

In Apama Studio, to specify the location of the Apama license file:

- In the Apama Studio menu bar, choose Window > Preferences.
- In the Preferences dialog that appears, click Apama.
- In the License file field, specify or navigate to the path of the Apama license file. The default path is `APAMA_WORK\license\license.txt`.
- Click Apply and then OK.

Overview of Developing Apama Applications

Chapter 2: Working with Projects

This section describes how to use Apama Studio to develop applications.

You can use Apama Studio to write Apama applications in the Apama Event Processing Language (EPL) and in the Apama in-process API for Java (JMon).

As you develop your application to a stage where you want to run it for testing purposes, you can launch it directly from Apama Studio. For more information on launching Apama projects, see "Launching Projects" on page 102.

Note: When using any of the Apama perspectives, you can redisplay the default perspective layout by selecting Window > Reset Perspective from the Apama Studio menu.

Creating Apama projects

To create an Apama project:

1. From the Apama Studio menu, select File > New > Apama Project to display the **New Project** dialog. If you are using the **Apama Workbench** perspective, you can also click the New Project button in top right of the **Workbench Project** view.
2. In the **New Apama Project** dialog, specify information for the following fields:
   a. In the Project Name field, enter the name of your new project. Project names typically use titlecase capitalization and may contain spaces, for example. “My Project”.

b. If you want to store the project in a directory other than the default location, clear the Use default location checkbox, click Browse, and navigate to and select the location for storing your new project.

3. If you do not need to change the bundles that will be included in the project click Finish which takes you to Step 6.

   Bundles are packages of Apama objects such as EPL files, event definition files, and event files that are required for specific types of applications. Adapter bundles contain the configuration files, service monitors, and other files associated with the standard Apama adapters.

   When you create a new project the settings for selected bundles remain the same as they were for the last time you created a new project. If you want to change these settings and specify other bundles for your application, click Next.

4. On the Configure new project page, add checkmarks to the bundles that are appropriate to the type of application you are developing,. For example if your application is a dataview application, select the DataView Service bundle.

   If you are creating a project for an EPL application, click Finish.

   If you are creating a project for a JMon application, add a check to the Add Java support to this project checkbox and click Next.

5. If you are creating a JMon application, in the **Configure the new Apama Java application** dialog, fill in the fields as desired. Click Finish.

6. If you are not currently in an Apama perspective, the **Open Associated Perspective?** dialog appears; it is up to you whether to select Remember my decision. Click Yes or No.

If you are in the **Apama Developer** perspective, Apama Studio displays the name of your new project in the **Project Explorer** view pane on the left of the perspective. If this is your first project, the other panes are blank.

Working with Projects

Adding resources to Apama projects

You specify the Apama resources that make up your application by adding them to the Apama project. Apama Studio recognizes and provides editing features for the following files:

- EPL files (`.mon`)
- Block definition files (`.bdf`)
- Function definition files (`.fdf`)
- Event files (`.evt`)
- JMon monitor and event files (`.java`)
- Scenario definition files (`.sdf`)
- Dashboard definition files (`.rtv`) — opens in Dashboard Builder

However, you can add any type of file to an Apama project and Eclipse uses the correct editor to open it.

When adding new blocks and functions to a project, you should create the definition files using Apama Studio.

Working with Projects

Creating new monitor files for EPL applications

EPL files define monitors and/or event types.

To add a new EPL file to an Apama project:

1. If you are in the **Apama Workbench** perspective, click the Show All Folders icon  if necessary to display the enhanced view that shows all the project's resources.
2. In the **Project Explorer** view or the **Workbench Project** view, right click the `monitors` folder of the project where you want to add the EPL file and select New > MonitorScript File. In the **Workbench Project** view you can also select the `monitors` folder and click the New MonitorScript File button 

3. In the **New MonitorScript File** wizard, enter information in the following fields:
   a. The Containing Folder field is the folder where the file will be saved; by default this is the folder of the currently selected project, but you can select another folder using the Browse button.
   b. In the File name field, specify the name of the new file. Specifying the `.mon` extension is optional as Apama Studio will add the `.mon` file extension. Apama Studio will not let you specify anything except `.mon` as a file extension.
   c. The Package field is optional; information in this field is an EPL package name.
4. Click Finish. The name of the new file now appears in the **Project Explorer** viewor the **Workbench Project** view under the project that contains it and the EPL file opens in the EPL editor.
5. In the EPL editor, add the desired EPL code and save the file.

For more information about EPL, see "Getting Started with Apama EPL" in *Developing Apama Applications in EPL*. For more information on the editing features available when writing EPL code, see "Editing Apama files".

Adding resources to Apama projects

Creating new event definition files for EPL applications

You can add new event definitions to an Apama Studio project. There are several ways to create event definitions:

- "Creating event definitions by adding EPL code" on page 32
- "Creating event definitions from XML files" on page 32
- "Creating event definitions from XSD files" on page 33

Creating event definitions by adding EPL code

To create an event definition by adding EPL code:

1. In the **Project Explorer** view, right-click the project's `eventdefinitions` folder and select New > Event Definition from the pop-up menu. The **New Event Definition** dialog is displayed.
2. In the **New Event Definition** dialog, in the Generate Event Definition using field, select EPL Editor and click Next. The **Choose output location for Event Definition** dialog is displayed.
3. In the **Choose output location for Event Definition** dialog, enter information in the following fields:
   a. The Containing Folder field is the folder where the file will be saved; by default this is the currently selected folder of the current project, but you can select another folder or project using the Browse button.
   b. In the File name field, specify the name of the new file. Specifying the `.mon` extension is optional as Apama Studio will add the `.mon` file extension. Apama Studio will not let you specify anything except `.mon` as a file extension.
   c. The Package field is optional; information in this field is an EPL package name.
4. Click Finish. The new event definition file is added to the specified folder in the project and the EPL file opens in the EPL editor.
5. In the EPL editor, add the desired EPL code and save the file.

Note, you can also create these types of event definitions by using the New > MonitorScript menu item; for details, see "Creating new monitor files for EPL applications" on page 31.

Creating event definitions from XML files

You can create an event definition that is based on the structure of an XML document. The generated event definition is based on the following:

- All fields are treated as `string`.
- All prefixes are ignored, in other words, event definitions will be generated even if inner elements have different namespaces.
- Namespace attributes (attributes having an `xmlns` prefix) are ignored.
- All inner event names are prepended with `Element_`.
- Generated event definitions will have an `xmlTextNode` field if the XML element contains a text value and also contains an attribute.

To create an event definition from an XML file:

1. In the **Project Explorer** view, right-click the project's `eventdefinitions` folder and select New > Event Definition from the pop-up menu. The **New Event Definition** dialog is displayed.
2. In the **New Event Definition** dialog, in the Generate Event Definition using field, select XML File and click Next. The **Choose output location for Event Definition** dialog is displayed.
3. In the **Choose output location for Event Definition** dialog, enter information in the following fields:

a. The Containing Folder field is the folder where the file will be saved; by default this is the currently selected folder of the current project, but you can select another folder or project using the Browse button.

b. In the File name field, specify the name of the new file. Specifying the `.mon` extension is optional as Apama Studio will add the `.mon` file extension. Apama Studio will not let you specify anything except `.mon` as a file extension.

c. The Package field is optional; information in this field is an EPL package name.

4. Click Next. This displays the **Select XML File** dialog.
5. In the **Select XML File** dialog, in the XML File field, specify the name of the XML file on which you want to base the event. You can use the Browse button to navigate to the desired file. The Down Arrow button switches the scope between Workplace and FileSystem.
6. In the **Select XML File** dialog, in the Event Name field, specify the name you want to assign to the root level event.
7. Click Finish. Apama Studio creates an EPL file that defines the root level event along with associated nested events.

Creating event definitions from XSD files

You can create event definitions that are based on the structures of elements defined in XSD schema files.

To create an event definition from an XSD file:

1. In the **Project Explorer** view, right-click the project's `eventdefinitions` folder and select New > Event Definition from the pop-up menu. The **New Event Definition** dialog is displayed.
2. In the **New Event Definition** dialog, in the Generate Event Definition using field, select XSD File and click Next. The **Choose output location for Event Definition** dialog is displayed.
3. In the **Choose output location for Event Definition** dialog, enter information in the following fields:

   a. The Containing Folder field is the folder where the file will be saved; by default this is the currently selected folder of the current project, but you can select another folder or project using the Browse button.

   b. In the File name field, specify the name of the new file. Specifying the `.mon` extension is optional as Apama Studio will add the `.mon` file extension. Apama Studio will not let you specify anything except `.mon` as a file extension.

   c. The Package field is optional; information in this field is an EPL package name.

4. Click Next. This displays the **Select XSD File** dialog.
5. In the **Select XSD File** dialog, click the Browse button to the right of the Schema Element/Type field. This displays the **Type Chooser** dialog.
6. In the **Type Chooser** dialog, specify the file that contains schema's global element that you want to use as the root element on which to base the event definition. Click the Browse button to navigate to the file. The Drop-down arrow allow you to change scope among Recent files, Local file system, Workspace, Remote URL, and XML Schema. After selecting a file, click OK. This returns to the **Select XSD File** dialog with the root element you selected displayed in the Schema Element/Type field.

7. In the **Select XSD File** dialog, in the Event Name field, specify the name you want to assign to the root level event.
8. Click Finish. Apama Studio creates an EPL file that defines the root level event along with associated nested events.

Creating new files for JMon applications

For Apama projects that use JMon applications, you can add new JMon applications, JMon monitors, and JMon events. In Apama applications written in Java, monitors and event types are implemented as Java classes.

Adding resources to Apama projects

Adding a new JMon application

To add a new JMon application to an Apama project:

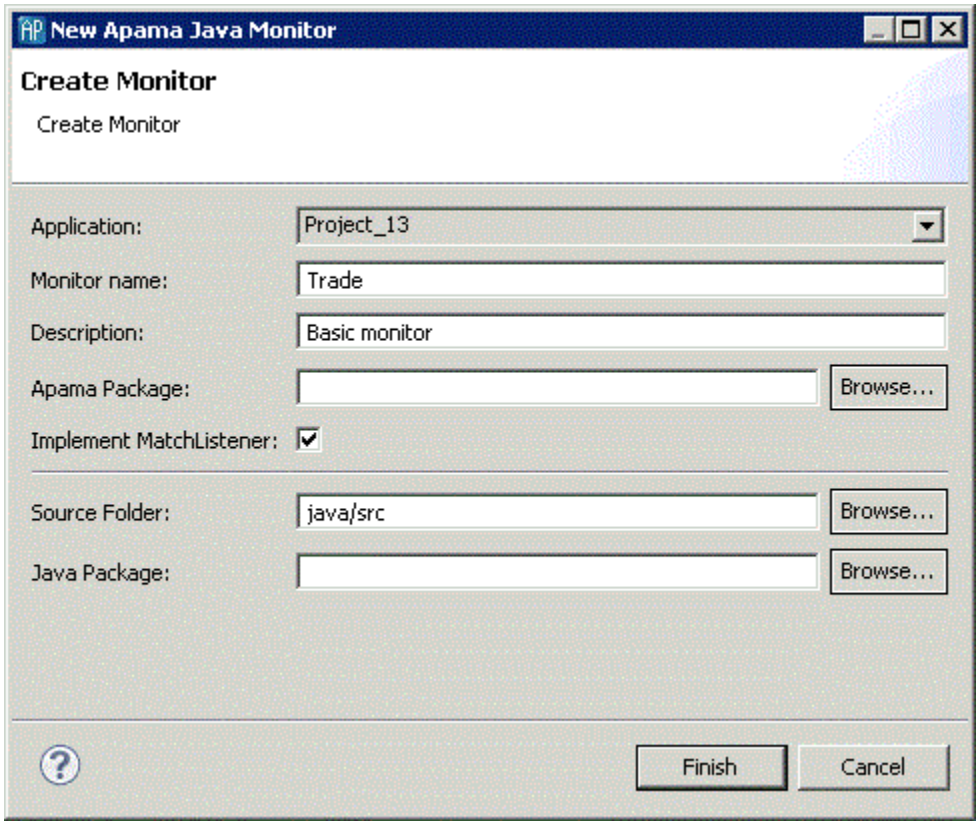
1. If you are in the **Apama Workbench** perspective, click the Show All Folders icon  if necessary to display the enhanced view that shows all the project's resources.
2. In the **Project Explorer** view or the **Workbench Project** view, right click the name of the project where you want to add the monitor and select New > Java Application. The **New Java Application** wizard is displayed.
3. In the **New Java Application** wizard, in the Configuration field specify the name of the application and click Finish. The application is added to the project and its configuration opens in the Apama Java configuration editor.
4. In the Apama Java configuration editor, specify the application's meta-data and add classes and contents as required. For more information on editing the configuration, see "Adding resources to JMon projects" on page 46.

Creating new files for JMon applications

Adding a JMon monitor

To add a new JMon monitor to an Apama project:

1. If you are in the **Apama Workbench** perspective, click the Show All Folders icon  if necessary to display the enhanced view that shows all the project's resources.
2. In the **Project Explorer** view or the **Workbench Project** view, right click the name of the project where you want to add the monitor and select New > Java Monitor. You can also select File > New > Other from the Apama Studio menu and then select Apama > Java Monitor from the **Select a wizard** dialog. The **New Apama Java Monitor** wizard is displayed.



3. In the **New Apama Java Monitor** wizard, enter information in the following fields:
   a. The Application field is the application to which you are adding the monitor.
   b. In the Monitor name field, specify the name of the new monitor. This will become the name of the class and the Java file.
   c. The Description field is optional.
   d. The Apama Package field is optional; this is the package of the monitor inside the correlator.
   e. Add a check to the Implement MatchListener check box if you want Apama Studio to generate the skeleton code for the class's `MatchListener` interface.
   f. The Source Folder field specifies the folder in the project to contain the file; by default this is `java/src`.
   g. The Java Package field is optional; this is the package of the created Java class.
4. Click Finish. The name of the new class file now appears in the **Project Explorer** viewor the **Workbench Project** view under the project that contains it and the `.java` file opens in the editor.

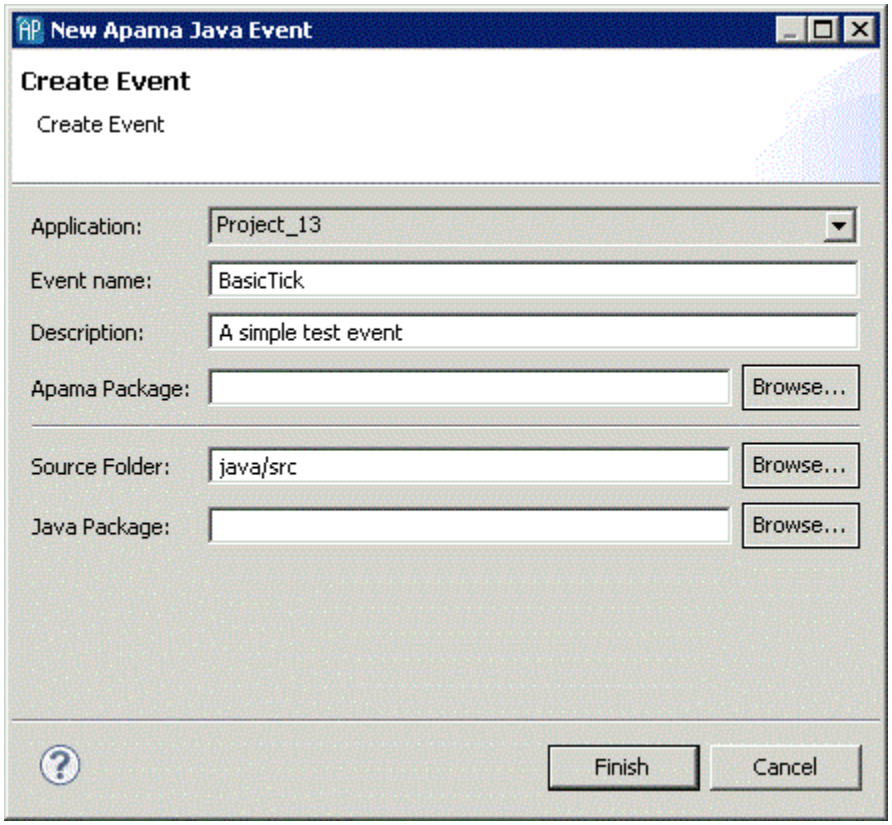
For more information about JMon applications, see "Overview of JMon Applications" in *Developing Apama Applications in Java*.

Creating new files for JMon applications

Adding a JMon event

To add a new JMon event to an Apama project:

1. If you are in the **Apama Workbench** perspective, click the Show All Folders icon  if necessary to display the enhanced view that shows all the project's resources.
2. In the **Project Explorer** view or the **Workbench Project** view, right click the name of the project where you want to add the monitor and select New > Java Event > Other. You can also select File > New from the Apama Studio menu and then select Apama > Java Event from the **Select a wizard** dialog. The **New Apama Java Event** wizard is displayed.



3. In the **New Apama Java Event** wizard, enter information in the following fields:
   a. The Application field is the application to which you are adding the event.
   b. In the Monitor name field, specify the name of the new event. This will become the name of the class and the Java file.
   c. The Description field is optional.
   d. The Apama Package field is optional; this is the package of the event inside the correlator.
   e. The Source Folder field specifies the folder in the project to contain the file; by default this is `java/src`.
   f. The Java Package field is optional; this is the package of the created Java class.
4. Click Finish. The name of the new class file now appears in the **Project Explorer** viewor the **Workbench Project** view under the project that contains it and the `.java` file opens in the editor.

For more information about JMon applications, see "Overview of JMon Applications" in *Developing Apama Applications in Java*.

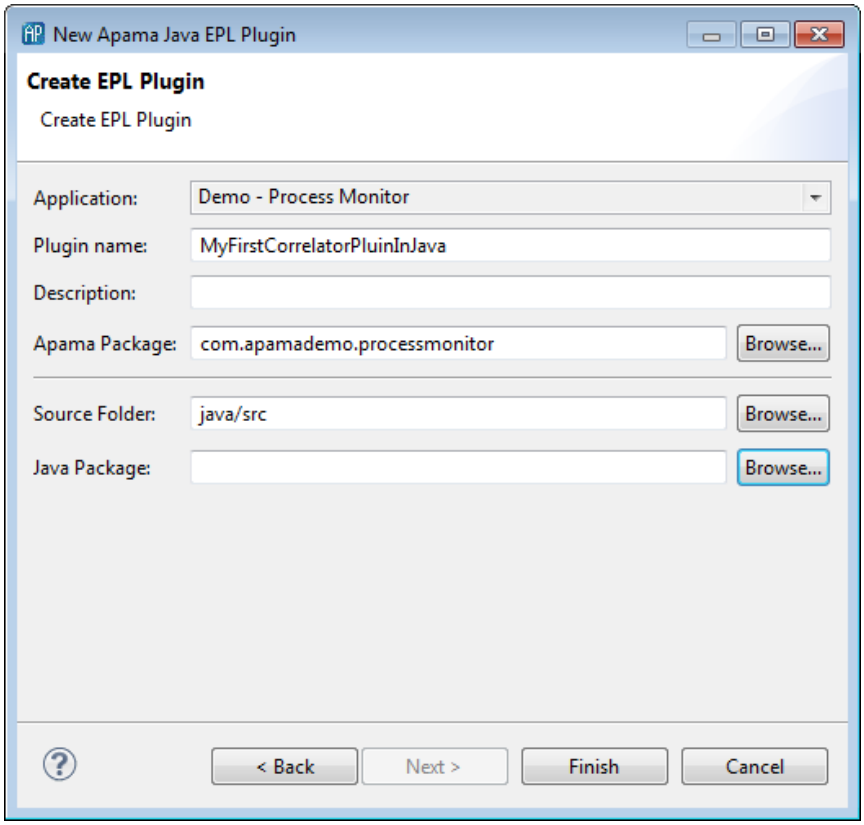
Creating new files for JMon applications

Adding an EPL Plugin written in Java

To add a new correlator plugin written in Java to an Apama project:

1. If you are in the **Apama Workbench** perspective, click the Show All Folders icon  if necessary to display the enhanced view that shows all the project's resources.

2. In the **Project Explorer** view or the **Workbench Project** view, right click the name of the project where you want to add the correlator plugin and select New > Java EPL Plugin. You can also select File > New > Other from the Apama Studio menu and then select Apama > Java EPL Plugin from the **Select a wizard** dialog. The **New Apama Java EPL Plugin** wizard appears.



3. In the **New Apama Java EPL Plugin** wizard, enter information in the following fields:

   a. The Application field is the application to which you are adding the plugin.

   b. In the Plugin name field, specify the name of the new plugin. This will become the name of the class and the Java file.

   c. The Description field is optional.

   d. The Apama Package field is optional; this is the package of the plugin inside the correlator.

   e. The Source Folder field specifies the folder in the project to contain the file; by default this is `java/src`.

   f. The Java Package field is optional; this is the package of the created Java class.

4. Click Finish. The name of the new class file now appears in the **Project Explorer** view or the **Workbench Project** view under the project that contains it and the `.java` file opens in the editor. In **Project Explorer**, the  icon indicates a Java plugin to the correlator.

5. Select the Show All Folders  icon in the Project View pane. You will find the source file for your Java class in the `java/src` package in your project. You can now add functionality to the class as required for your application. This can be called from EPL using the mechanism described in "Using Java plug-ins" in *Writing Correlator Plugins*.

For applications that you plan to inject into a correlator, the recommendation is to create separate jar files for:

- EPL plugins written in Java
- JMon applications

Although the mechanism for creating these jars and describing their meta-data is similar, the interactions of these two different uses of injected jars mean that they will often need to be injected into the correlator separately. The creation of separate jar files ensures that you can inject your application components in the correct order, which is typically:

1. EPL plugins written in Java
2. EPL monitors and events
3. JMon applications

For more information, see "Overview of JMon Applications" in *Developing Apama Applications in Java* and "Writing Correlator Plugins in Java" in *Writing Correlator Plugins*.

Creating new files for JMon applications

Creating new scenarios

Scenarios define templates that, when instantiated with a set of parameters supplied by the user, implement an application's strategy. To add a new scenario to an Apama project:

1. In the **Project Explorer** view or the **Workbench Project** view, right click the `scenarios` folder of the project where you want to add the scenario and select New > Scenario. In the **Workbench Project** view you can also select the `scenarios` folder and click the New Scenario button 

2. In the **New Scenario** wizard, enter information in the following fields:
   a. The Containing Folder field is the folder where the definition file will be saved; by default this is the folder of the currently selected project, but you can select another folder using the Browse button.
   b. In the File name field, specify the name of the new file. Specifying the `.sdf` extension is optional as Apama Studio will add the `.sdf` file extension. Apama Studio will not let you specify anything except `.sdf` as a file extension.
3. Click Finish. Apama Studio adds the name of the new scenario definition file to project and opens the new scenario in the Apama Event Modeler.

Using Event Modeler, you complete the scenario by adding state and rules and specify the blocks and functions necessary to implement the desired strategy. For more details on developing scenarios, see "Overview of Using Event Modeler" in *Developing Apama Applications in Event Modeler*.

Adding resources to Apama projects

Creating new blocks

This section is a brief description of how to create a new block in Apama Studio. For more details on this process, see "Creating Blocks" on page 81. You can create a block in Apama Studio either from scratch or from an existing event definition.

For information on creating new blocks, see the following topics:

- "Creating a block with the block editor" on page 39
- "Creating a block from an EPL event definition" on page 40
- "Adding EPL code to a block" on page 41

Note: When you create a new block, you should place it in the project's default `blocks` directory. This directory is found in the project's `catalogs` directory. The `blocks` directory has a name in the form *`project_name`* `blocks`. So, for example, the default block directory of a project named `My_Project` will be `catalogs\My_Project blocks`. If you place the block in the default block directory, scenarios created in the project will automatically find them and make them available in Event Modeler when you are displaying the scenario.

Creating a block with the block editor

Creating a block in Apama Studio consists of two main steps. In the first step you create the block metadata and specify its interface. In the second step you add the EPL code that implements the block's behavior.

To add a new block to an Apama project using the block editor:

1. In the **Project Explorer** or **Workbench Project** view, right-click the default block folder (typically `catalogs\`*`project_name`* `blocks`) and select New > Block button from the pop-up context menu. The **New Event Modeler Block** dialog is displayed.
2. In the **New Event Modeler Block** dialog, specify the name of the project in the Containing folder field if you want to create the block in a different project; you can also click Browse to select the project from a list of projects in the workspace.
3. Also in the **New Event Modeler Block** dialog, select Block Editor in the Generate block using field and click Next. The **Create a new Apama Block** dialog is displayed.
4. In the **Create a new Apama Block** dialog, fill in the various fields or accept the default values. The most important fields to set at this point are the block name and version. The Type field specifies what type of code will be generated for the block. The choices are:
   - Callback — Implements the block as an event type; this is the default. Generates EPL code to which you can add custom code.
   - Callback (DEBUG) — Implements the block as an event type. Generates EPL code to which you can add custom code. Also generates statements that can help you debug the EPL you are adding. You can easily switch your block between Callback and Callback (DEBUG).
5. Click Finish. The block definition file for the new block is added to the project and the block's metadata is displayed in the Builder tab of the block editor.

6. The left side of the Builder tab displays the parameters, input feeds, output feeds, and operations that make up the block; a new block will not contain any entries here. To add one of these elements:

    a. Right click the element you want to add and select Add Parameter, Add Input Feed, Add Output Feed, or Add Operation. The right side of the Builder tab displays the item's properties.

    b. Fill in the values for the properties.

7. For input feeds and output feeds, right click the element and select Add Field.

8. In the Properties panel for the field you added in the previous step fill in the values for the properties and field validation specifications. Repeat for each field you want to add to an input or output feed.

9. When you finish specifying the block's metadata, save the file.

For details about adding EPL code to a block, see "Adding EPL code to a block" on page 41.

Creating a block from an EPL event definition

In cases where you want to create an event from a scenario or to update the scenario types from an Apama event, you should create a block based on the event type.

Creating a block from an existing EPL event definition consists of selecting the event definition from which you want to create the block, specifying whether you want to create an input block or an output block, and specifying what fields in the event are to be used when the block is generated. For input blocks, the specified fields are used to construct listeners; for output blocks, the specified fields are used to set the values of output parameters.

To add a new block to an Apama project from an existing EPL event definition:

1. In the **Project Explorer** or **Workbench Project** view, right-click the default block folder (typically `catalogs\project_name\blocks`) and select New > Block from the pop-up context menu. The **New Event Modeler Block** dialog is displayed.

2. In the **New Event Modeler Block** dialog, specify the name of the project in the Containing folder field if you want to use an event definition in a different project; you can also click Browse to select the project from a list of projects in the workspace.

3. Also in the **New Event Modeler Block** dialog, select EPL event definition in the Generate block using field and click Next. The **Select the Block type and Event type** dialog is displayed.

4. In the **Select the Block type and Event type** dialog, click Browse to display the **Event Type Selection** dialog. Note, that events with field types chunk, listener, or stream are not listed because they cannot be emitted, routed, or enqueued.

    a. In the **Event Type Selection** dialog's Choose an Event Type field, enter the name of the event. As you type, event types that match what you enter are shown in the Matching Events list.

    b. In the Matching Events list, select the name of the event type you want to use to generate the block. The name of the EPL file that defines the selected event is displayed in the status area at the bottom of the dialog.

    c. Click OK.

5. In the **Select the Block type and EPL type** dialog, specify whether you want to generate an input block or an output block. Also, in the Include comments field, add a check if you want to include ApamaDoc comments from the event definition in the generated block code and click Next. The **Create New Apama Block** dialog is displayed.

6. In the **Create a New Apama Block** dialog, specify the name, version, type and description of the new block. By default, Apama Studio uses the name of the event as the basis for the name of the block. The Type field specifies what type of code will be generated for the block. The choices are:

   - Callback — Implements the block as an event type; this is the default. Generates EPL code to which you can add custom code.
   - Callback (DEBUG) — Implements the block as an event type. Generates EPL code to which you can add custom code. Also generates statements that can help you debug the EPL you are adding. You can easily switch your block between Callback and Callback (DEBUG).

   When you finish entering information, click Next. The **Select Block Parameters from Event Fields** dialog is displayed.

7. In the **Select Block Parameters from Event Fields** dialog, in the Event Fields field, specify the fields to be used to create the block.

   Note, for input blocks, fields with inner event fields, and fields of `dictionary` or `sequence` types are not displayed. Also, for both input and output blocks, `constant`, `action`, and `context` fields are not displayed.

   - When you create a new input block, code is generated for a block parameter and listener for a specified field. If you select more than one field, block parameters and listeners are created for all combinations of all the selected fields. The maximun number of fields you can select is ten.

     For input blocks, code is also generated for an output feed that includes all the fields in the specified event, as well as `start` and `stop` actions that activate and deactivate the listeners.
   - When you create a new output block, code is generated for parameters for the event fields you specify. This means that values for those fields can be specified when the block is used.

8. Click Finish.

Apama Studio creates the block's `.bdf` file (the block definition file) and opens it in the block editor. The left side of the Builder tab displays the parameters, input feeds, output feeds, and operations that are automatically generated from the information you specified for the block. For details about adding EPL code to a block, see "Adding EPL code to a block" on page 41.

Adding EPL code to a block

When you save a block, Apama Studio generates the underlying code that defines the block's interface and saves it as a *block definition file* with a `.bdf` extension. To this file, you then add EPL code to implement the necessary behavior. To add code to the block:

1. If necessary, double-click the block in the **Project Explorer** to open it in the block editor. In the block editor display the Source tab.
2. On the Source tab, code is displayed either with a gray background or a white background. Code with a gray background is maintained by Apama Studio and is not editable. The sections of code with a white background are the areas where you add your custom EPL code.

For more information on adding EPL code to a block, see "Adding EPL code to the block definition" on page 85. Also, for background information on the elements that make up the actual code of a block's block definition file, see "File Definition Formats" on page 188.

Creating new scenario functions

Many Apama applications are implemented partially or entirely as scenarios. Apama Studio supplies many pre-packaged scenario functions that can be used in conditions or actions when defining rules for scenarios. In addition, you can define your own scenario functions within Apama Studio.

Note: When you create a new function, you should place it in the project's default functions directory. This directory is found in the project's `catalogs` directory. The function directory has a name in the form *`project_name`* `functions`. So, for example, the default `functions` directory of a project named `My_Project` will be `catalogs\My_Project functions`. If you place the function in the default function directory, scenarios created in the project will automatically find them and they will be available in Event Modeler when you are displaying the scenario.

To define a new scenario function:

1. If you are in the **Apama Workbench** perspective, click the Show All Folders icon  if necessary to display the enhanced view that shows all the project's resources.
2. In the Project Explorer view or the **Workbench Project** view, open the `catalogs` folder of the project where you want to add the EPL file.
3. Right click the functions folder within the `catalogs` folder. It will have a name such as `MyProject functions`. Select New > Scenario Function.

   In the **Workbench Project** view you can also select the functions folder and click the New Scenario Function button 

   .
4. In the **New Scenario Function** wizard, specify information for the following fields:
   a. The Containing Folder field is the folder where the function definition file will be saved; by default this is the functions folder you selected in Step 4, but you can select another folder using the Browse button.
   b. In the Function Name field enter a unique name for the function. By convention, this should be in uppercase.
   c. In the Display Name field enter the name that will be displayed in the Apama Event Modeler tool. By default this the same as the Function Name but you can change it if you like; it does not need to be unique.
   d. In the Return type field specify the type for the value returned by the function.
   e. In the Description field specify the descriptive text that will accompany the function in the Apama Event Modeler tool.
5. Click Finish. This generates the definition file for the function and opens the file in the Apama editor.
6. The function definition file is an XML file. To implement the behavior of the function you need add the EPL code to the `action` statement in the file's `<code>` element. In the `.fdf` file, this section is labeled:

   ```
   // TODO: put MonitorScript code to implement the function here
   ```

After you have added a new scenario function to a project, when you look at a scenario in the project with the Apama Event Modeler, the new function will be included in the Event Modeler's function catalog associated with the project.

For more information on using functions in scenarios, see "Using functions in Event Modeler" in *Developing Apama Applications in Event Modeler*. For more information on the XML format of the function definition file, see "File Definition Formats" on page 188.

Adding resources to Apama projects

Creating new dashboards

Dashboards provide the ability to view and interact with scenarios and DataViews. They contain charts and other objects that dynamically visualize the values of scenario variables and DataView fields. Dashboards can also contain control objects for creating, editing, and deleting scenario instances and DataView items. Normally, you add new dashboards after you have substantially developed and tested your application.

You can create dashboards with either the **Dashboard Generation** wizard or the Dashboard Builder. The wizard allows you to generate simple, default dashboards, customized with your choices regarding basic layout and visualization objects to use. The Builder is a graphical composition tool that gives you fine-grained control over a dashboard's appearance and behavior.

The following sections describe how to create dashboards to your project:

- "Creating dashboards with the Dashboard Generation wizard" on page 43
- "Creating dashboards with the Dashboard Builder" on page 44

Adding resources to Apama projects

Creating dashboards with the Dashboard Generation wizard

To create dashboards that are generated by the **Dashboard Generation** wizard, follow these steps:

1. If you are using the **Project Explorer** view, ensure that a project is selected.
2. In either the **Project Explorer** view or the **Workbench Project** view, select New > Dashboard from the File menu. (You can also right-click in the navigation pane and select Dashboard from the popup menu. In the **Workbench Project** view, you can also click the New button that is above the navigation pane and select Dashboard from the Apama folder, or click the down arrow that is next to the New button, and select Dashboard from the popup menu.)
3. In the **New Dashboard** dialog, click the **Dashboard Generation** wizard radio button, and enter a name in the Configuration name field. The **Dashboard Generation** wizard uses this as the name of the new configuration that will be used to generate your Dashboards. When you use the Wizard, you can accept a default configuration or specify a custom configuration.
4. Click Finish.The Dashboard Generation Wizard appears, and displays the new dashboard-generation configuration. In addition, a new dashboard-generation configuration file (`dashboard_generation.xml`) appears under the current project's config folder, if one wasn't already present.

For information on using the **Dashboard Generation** wizard, see "Generating Dashboards" on page 168, in *Using Apama Studio*.

Adding resources to Apama projects

Creating dashboards with the Dashboard Builder

To create a dashboard and develop it with the Dashboard Builder, follow these steps:

1. If you are using the **Project Explorer** view, ensure that a project is selected.
2. In either the **Project Explorer** view or the **Workbench Project** view, select New > Dashboard from the File menu. (You can also right-click in the navigation pane and select Dashboard from the popup menu. In the **Workbench Project** view, you can also click the New button and select Dashboard from the Apama folder, or click the down arrow next to the New button, and select Dashboard from the popup menu.)
3. In the **New Dashboard** dialog, click the Dashboard Builder radio button, and enter information in the following fields:
   a. Containing Folder is the folder where the dashboard definition file will be saved; by default this is the dashboards folder of the current project, but you can select another folder using the Browse button.
   b. File name specifies the name of the new definition file. Specifying the `.rtv` extension is optional as Apama Studio will add the `.rtv` file extension. Apama Studio will not let you specify anything except `.rtv` as a file extension.
4. Click Finish. The name of the new dashboard definition file appears under the current project's dashboards folder. Additionally, in a separate window Apama Studio opens the Apama Dashboard Builder tool showing the new dashboard.

Using Dashboard Builder, you complete the dashboard by adding visualizing and control objects and connecting them to live correlator data. For more information on completing dashboards, see Chapter 1, "Introduction" in *Building Dashboards*.

Adding resources to Apama projects

Creating new dashboard-deployment configurations

Dashboard-deployment configurations contain the information necessary for the generation of deployment packages for a project's dashboards. Follow these steps to create a deployment configuration:

1. If you are using the **Project Explorer** view, ensure that a project is selected.
2. In either the **Project Explorer** view or the **Workbench Project** view, select New > Dashboard Deployment from the File menu. (You can also right-click in the navigation pane and select Dashboard Deployment from the popup menu. In the **Workbench Project** view, you can also click the New button that is above the navigation pane and select Dashboard Deployment from the Apama folder, or click the down arrow that is next to the New button, and select Dashboard Deployment from the popup menu.)
3. In the **New Dashboard Deployment Configuration** dialog, enter a name in the Configuration field. The Dashboard Deployment Configuration Editor uses this as the name of the new configuration.

4. Click Finish.The Dashboard Deployment Configuration Editor appears, and displays the new dashboard configuration. In addition, a new dashboard-deployment configuration file (`dashboard_deploy.xml`) appears under the current project's config folder, if one wasn't already present.

For information on using the Dashboard Configuration Deployment Editor, see "Using the Deployment Configuration editor" on page 181 in "Generating Dashboards" on page 168 in *Using Apama Studio*.

Adding resources to Apama projects

Creating new event files

To add a new event file to an Apama project:

Event files are used to supply events of a specific type. For example in applications written in EPL, you should usually send in an event to tell the monitors making up the application to start listening for events once everything has been injected.

1. If you are in the **Apama Workbench** perspective, click the Show All Folders icon  if necessary to display the enhanced view that shows all the project's resources.
2. In the **Project Explorer** view or the **Workbench Project** view, right click the `events` folder of the project where you want to add the event file and select NewFile > Event. In the **Workbench Project** view you can also select the `events` folder and click the New Event File button  .
3. In the **New Event File** wizard, enter information in the following fields:
   a. The Containing Folder field is the folder where the file will be saved; by default this is the folder of the currently selected project, but you can select another folder using the Browse button.
   b. In the File name field, specify the name of the new file. Specifying the `.evt` extension is optional as Apama Studio will add the `.evt` file extension. Apama Studio will not let you specify anything except `.evt` as a file extension.
4. Click Finish. The name of the new file now appears in the **Project Explorer** viewor the **Workbench Project** view under the project that contains it and the event file opens in the Apama Studio event editor.

Adding resources to Apama projects

Adding resources to EPL projects

To add an existing EPL file that defines a monitor to an Apama project, you add the file as an External Dependency. You can also add EPL files that are contained in an Apama Correlator Deployment Package file. To add a file to an active project as an external dependancy:

1. Select Project > Properties from the Apama Studio menu.
2. In the **Properties** dialog, select MonitorScript Build Path from the left hand pane and select the External Dependencies tab in the right pane.

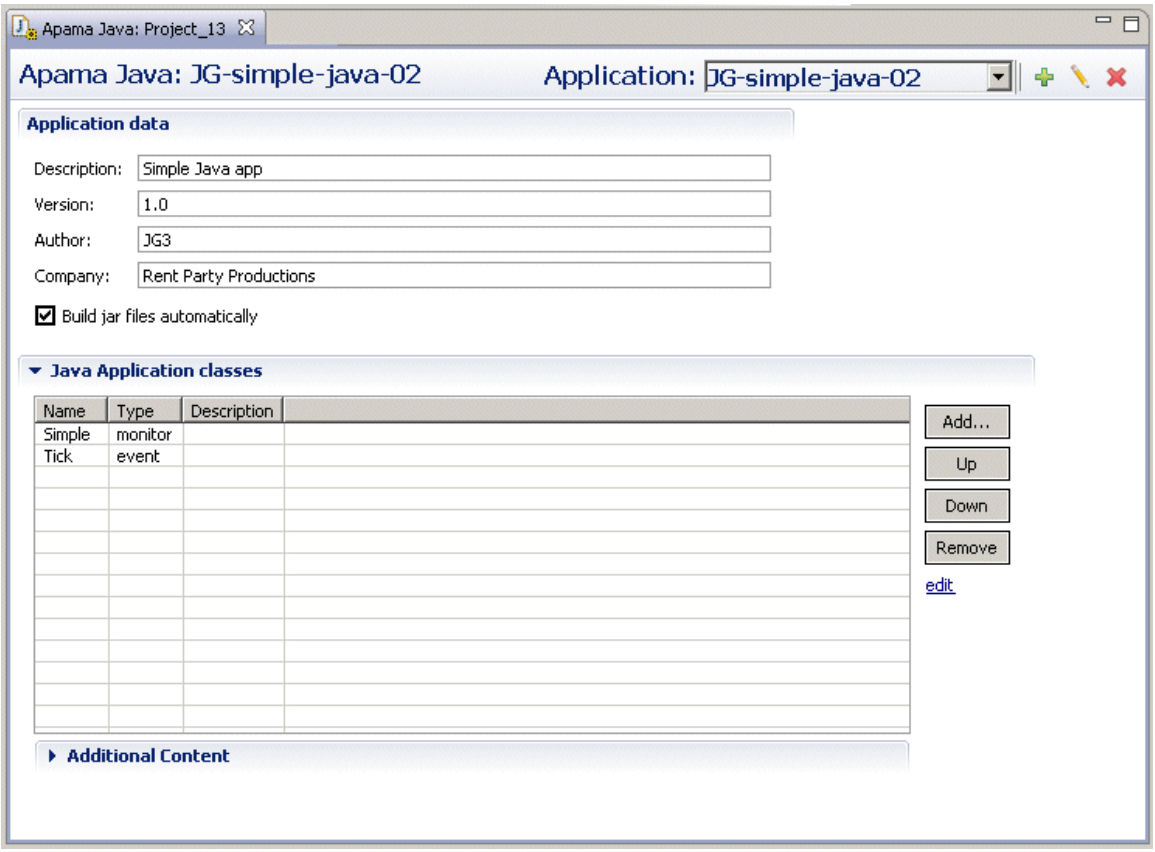
3. Click Add External button and navigate to the files you want to add. You can select EPL (`.mon`) files or Correlator Deployment Package (`.cdp`) files.
4. When finished, click OK at the **Properties** dialog.

Adding resources to Apama projects

Adding resources to JMon projects

To add existing resources to a JMon project, you modify the project's configuration. The configuration information is stored in the project's `config/apama_java.xml` file.

To add resources to a JMon project, double click the project's `config/apama_java.xml` file. The Apama Java configuration editor opens.



The Application drop-down list on the right of the editor's title bar shows the application you are editing. You can select any JMon application currently included in the project and you can add JMon applications from other projects to your current project.

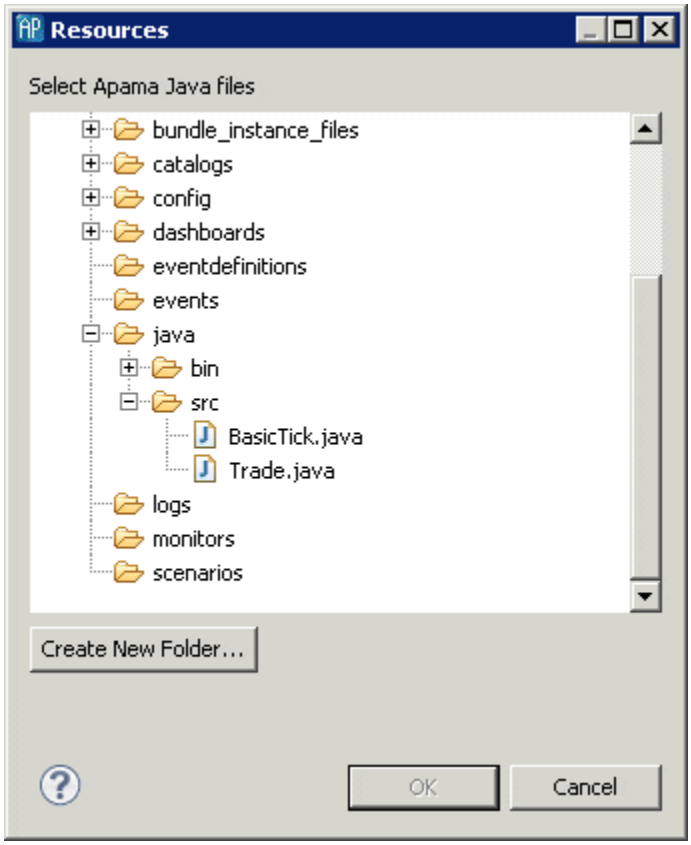
You can add other JMon classes (JMon events and JMon monitors) and non-JMon Java files to a project.

Adding resources to Apama projects

Adding JMon applications

To add a JMon application from another Apama project:

1. Click the Add application button ( ). The **New Application Name** dialog is displayed.
2. Specify a name for the new application and click OK. The editor will display the settings for this new application and the name will be added to the editor's drop down list.
3. In the Apama Java configuration editor, expand the Java Application Classes heading. This displays the current list of the Java classes included in the project. Click Add; this displays the **Select Java Apama Files** dialog showing the available JMon Application projects.



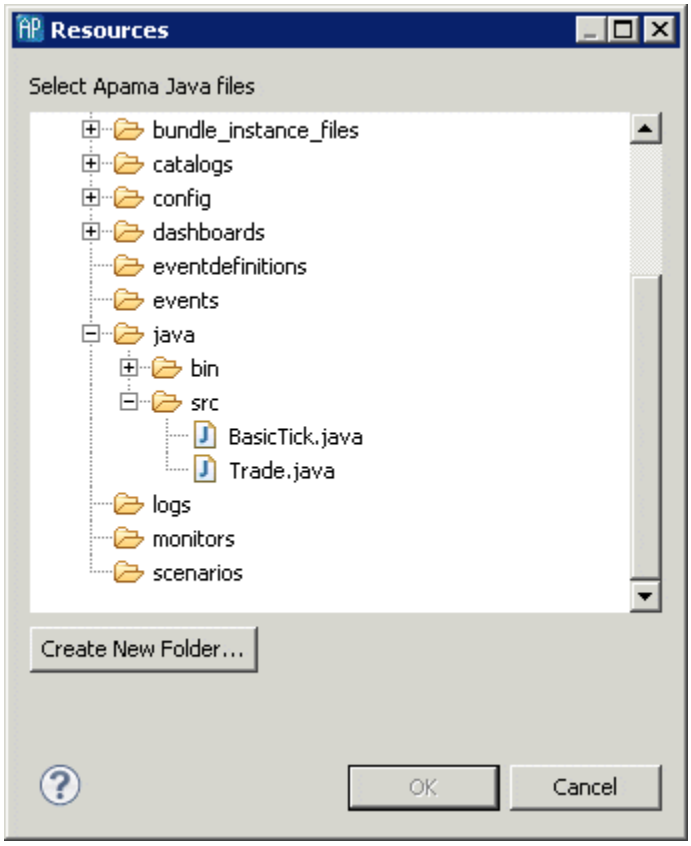
4. Select the JMon application file you want and click OK.
5. When you save the configuration information, Apama Studio generates a .jar file and puts it in a new folder in the current project.

Adding resources to JMon projects

Adding JMon classes

In order to add a JMon monitor or event from another Apama project:

1. In the Apama Java configuration editor, expand the Java Application Classes heading. This displays the current list of the Java classes included in the project. Click Add; this displays the **Select Java Apama Files** dialog showing the available JMon application projects.



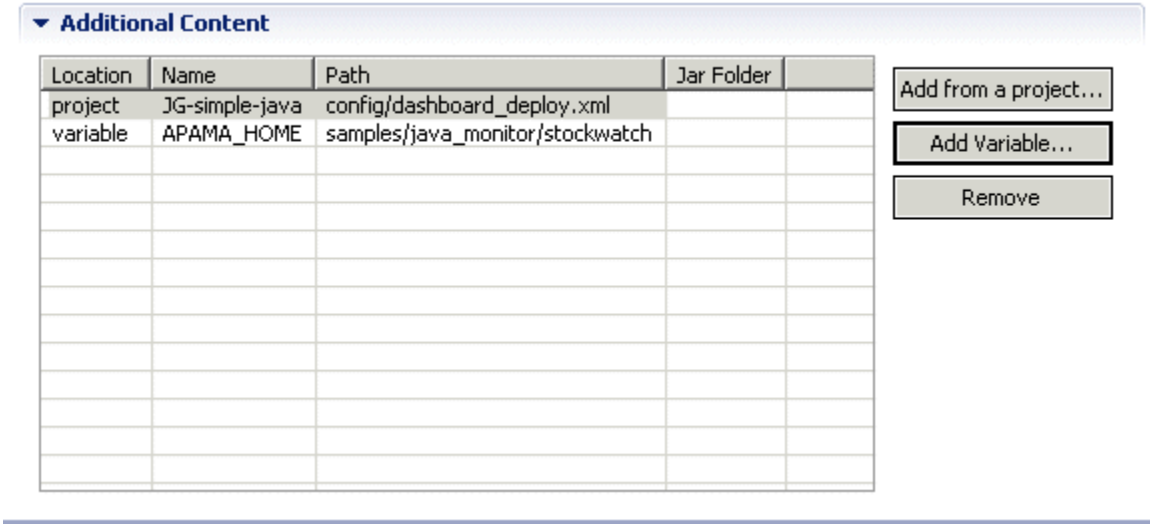
2. In the **Select Java Apama Files** dialog navigate to the class you want to add, select it, and click OK.
3. In the Apama Java configuration editor, you can also change the order in which the JMon classes are listed with the Up and Down buttons. This affects the way the files are ordered in the manifest file and the order in which they are injected in the correlator. Events are injected first, followed by monitors; in both cases, they are injected in the order they are listed here.

Adding resources to JMon projects

Adding non-JMon Java files

To add non-JMon Java files:

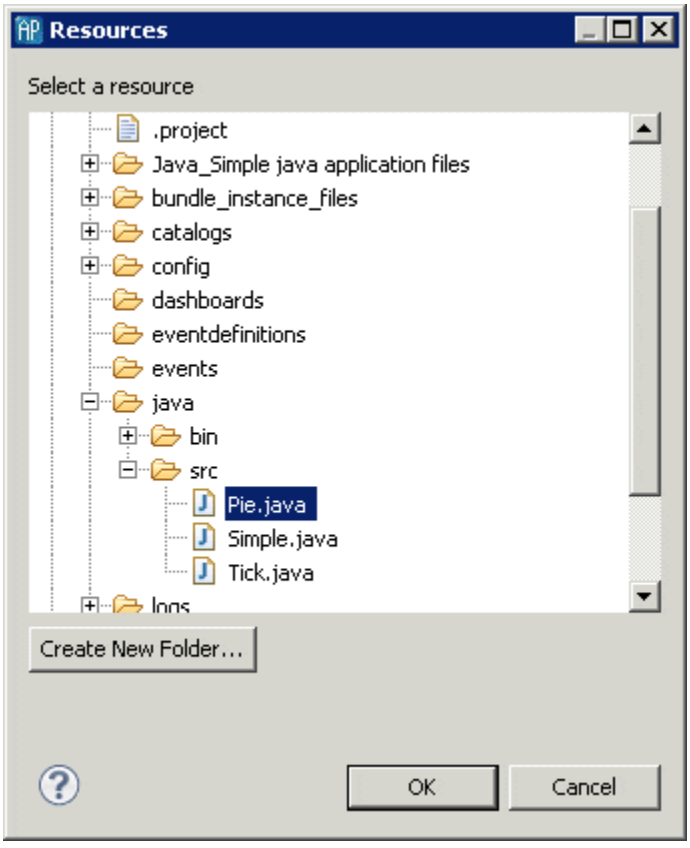
1. In the Apama Java configuration editor, expand the Additional Content heading. This displays the current list of non-JMon Java files included in the project.



You can add files from other Apama projects and you can add non-project files from outside the Apama Studio workplace.

2. To add a file from an existing Apama project:

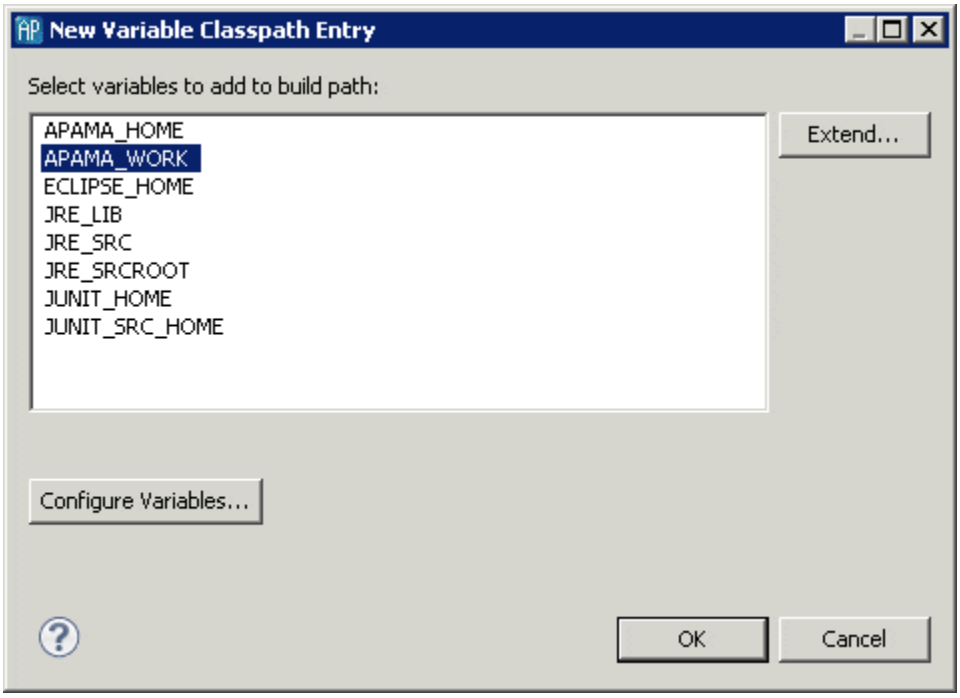
   a. Click Add from a project. This displays the **Select a Resource** dialog.



   b. Navigate to the file you want to add, select it, and click OK.

3. To add a file from outside Apama Studio:

   a. Click Add a Variable. The **Select variables to add to build path** dialog is displayed



.

b. Select a variable and, if necessary, click Extend to identify specific folders on the path. Click OK.

Adding resources to JMon projects

Adding bundles to projects

Bundles are named collections that group the Apama objects that are necessary for different types of applications. *Standard Bundles* contain service monitors and event definitions that are specific to the type of application you are building. For example, applications that include scenarios need the Scenario Service bundle, while applications that use data views need the DataView bundle. *Adapter Bundles* contain adapter configuration files, monitors, and other files needed by the Integration Adapter Framework (IAF). For information on adding adapters, see "Adding adapters to projects" on page 53.

Adding resources to Apama projects

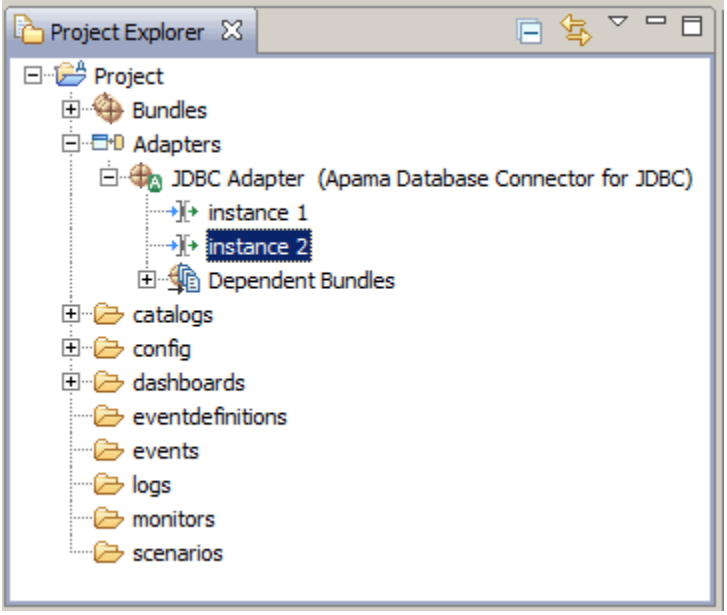
Bundle instances

Many bundles and adapters contain only EPL or Java files; adding these bundles means that references to the bundle files will be added to the project build path, but does not involve copying any files into the project. However, when you add a bundle that contains customizable files such as `.evt` or adapter configuration files, Apama Studio physically copies these files into the project, in addition to the changes made to the project build path.

If you add more than one instance of a particular bundle or adapter to the project, Apama Studio creates separate copies of these files for each instance (except for bundles where it only makes sense to have one copy of the instance files per project). If a bundle contains instance files, you can change and customize those files in any manner. It is possible to add multiple instances of such bundles to a given project, to allow different customizations of the same instance files, for example if your

application needs to run with two instances of a particular adapter, each connected to a different data provider.

If a bundle contains instance files, the **Add Bundle Instance** dialog will prompt for a unique *bundle instance name* to identify the instance, to distinguish it from any other instances of the same bundle. This instance name will be included in the filename of the bundle instance files when they are copied into the project. If only one instance of a given bundle will be required, it is fine to use the default bundle instance name. Each bundle or adapter instance is represented in the **Project Explorer**.

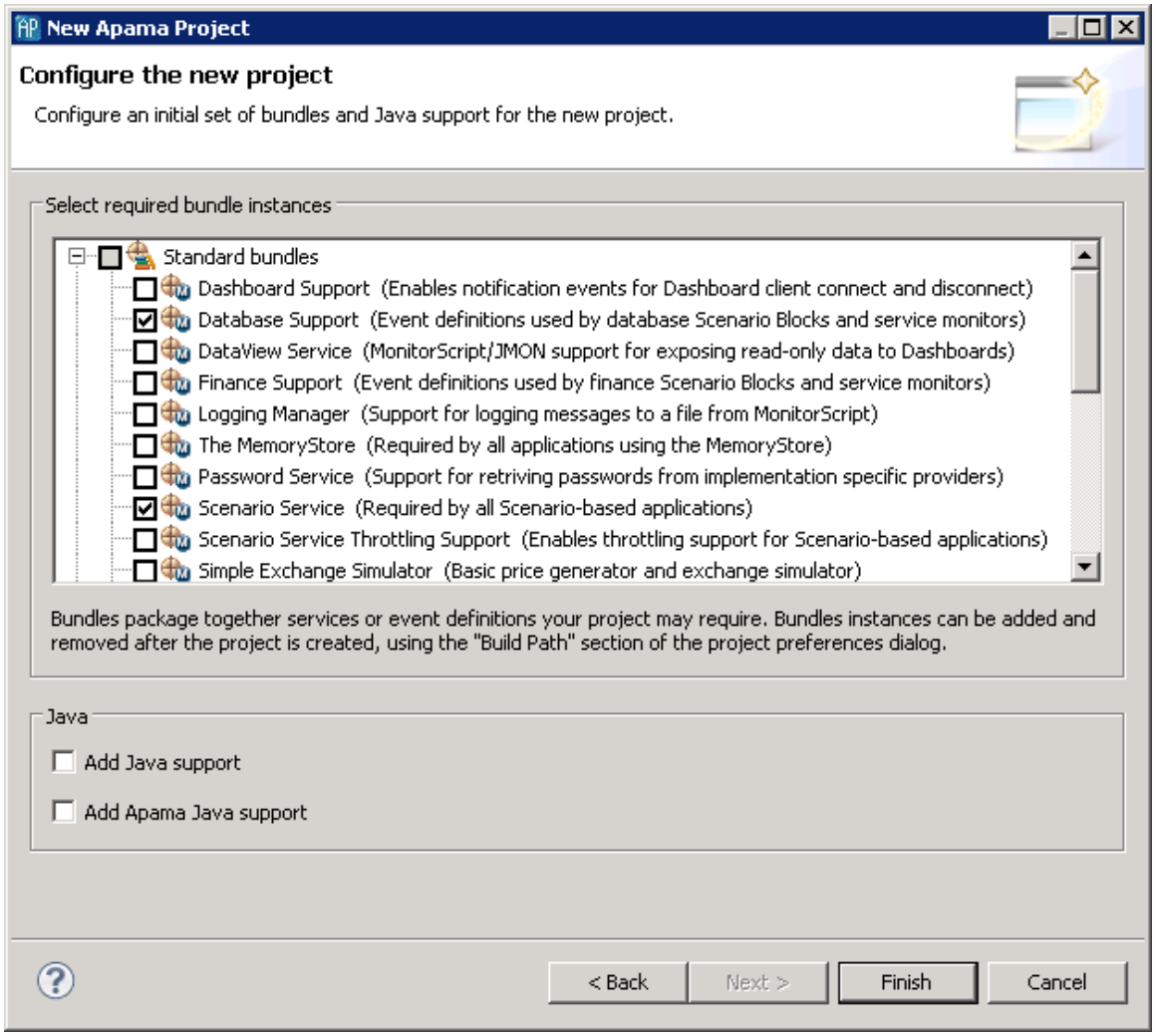


After adding a bundle that contains instance files, the instances can be removed or updated independently (right-click the instance name and select Remove Instance from the pop-up menu). After a bundle with instance files is removed, a dialog will appear to confirm whether you also wish to delete the instance files that were copied to the project. Usually you will want to delete these files at the same time as removing the bundle, as long as backup copies of any important customizations have been made.

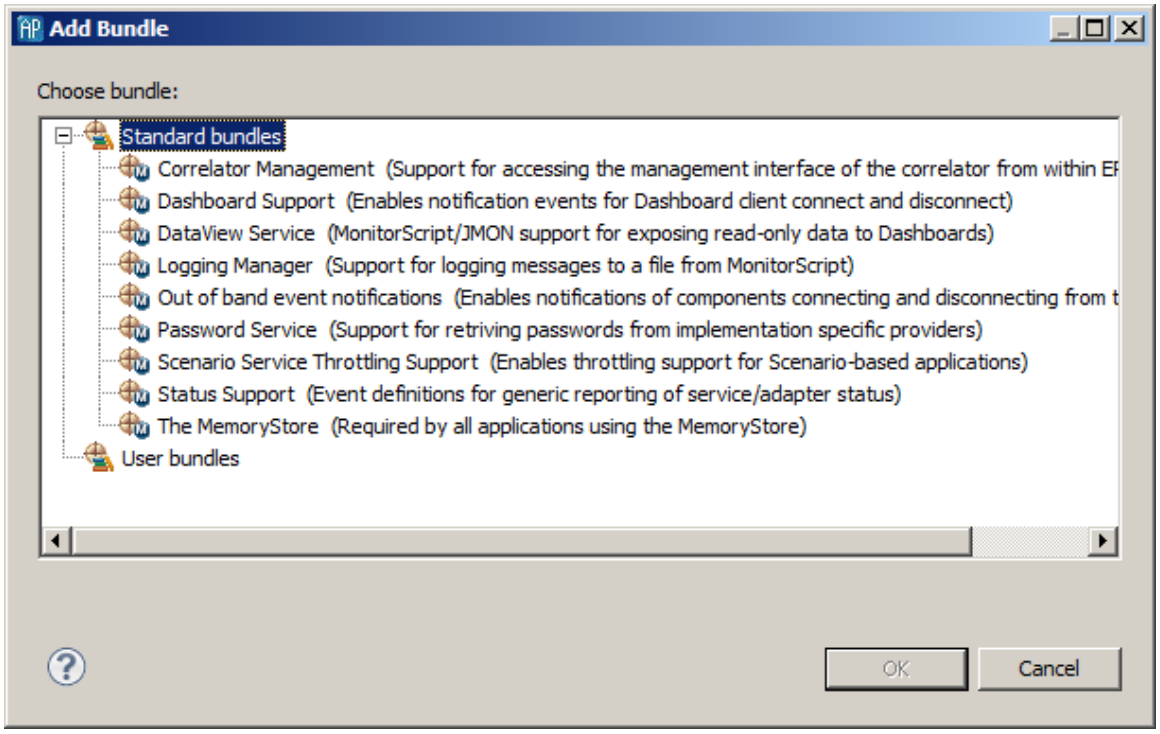
It is never necessary to have more than one instance of a bundle that does not contain any instance files, so this option is disabled in the **Add Bundle Instance** dialog.

To add a bundle to an Apama project:

1. There are two ways of adding a bundle to a project.
   - *If you are creating a new Apama project*, select File Project > New > Apama , give it a name, and click Next. The New Apama Project dialog opens.



- *If you are adding a bundle to an existing project,* in the Project Explorer right-click the project and select Apama > Add Bundle. The **Add Bundle** dialog opens.



2. Select the bundle that is appropriate for your application. Click Finish or OK.

Apama Studio adds the bundle to the `Bundles` folder in your project along with the supporting monitors such as `DatabaseSupport.mon`. If a bundle contains a Correlator Deployment Package you can see this in the bundle's hierarchy in the project's `Bundles` folder in Project Explorer.

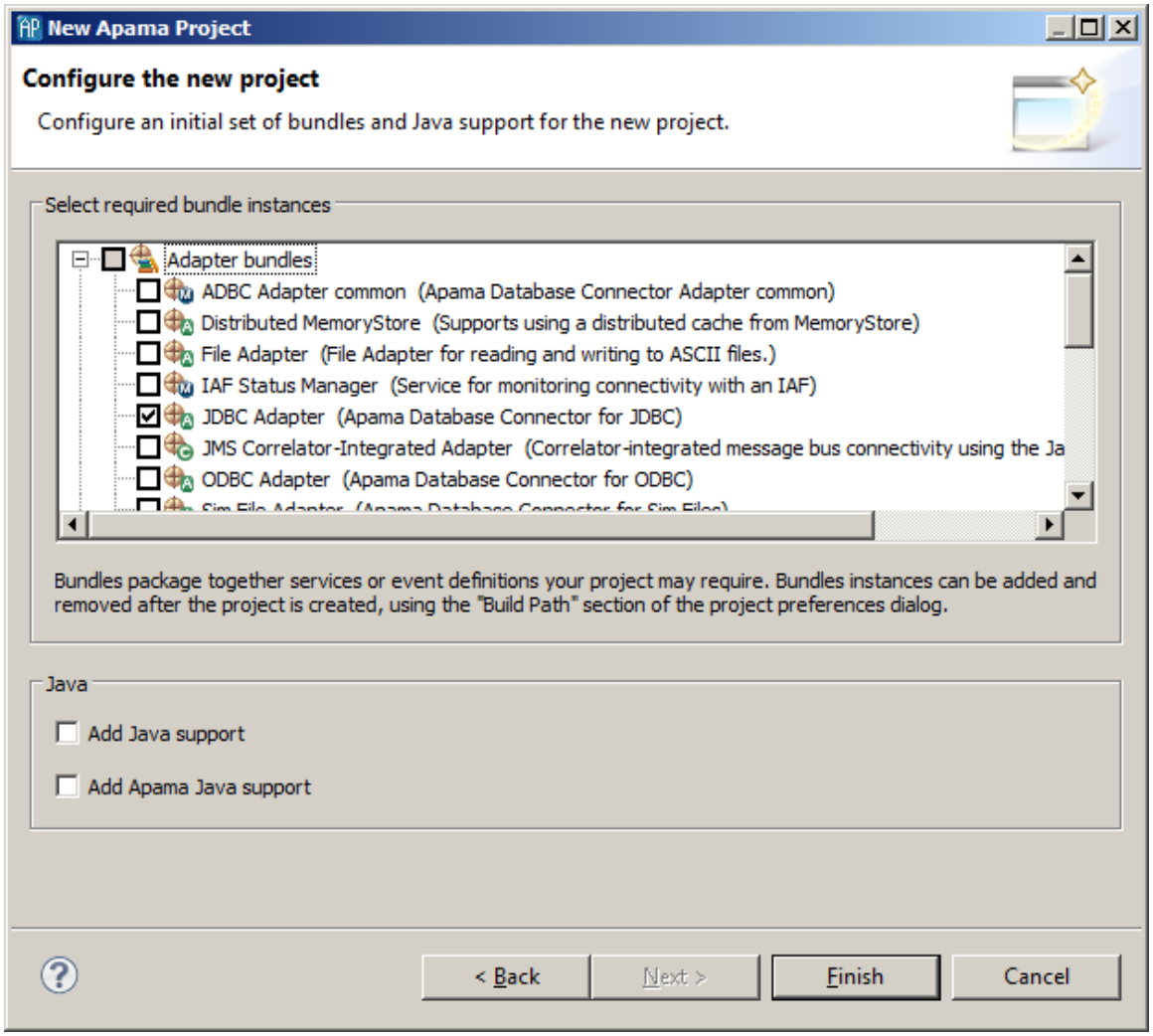
Adding bundles to projects

Adding adapters to projects

Apama provides standard adapters that communicate with third-party messaging systems, transforming incoming messages into Apama events and, in the opposite direction, transforming Apama events into the proprietary representations required by third-party messaging systems. In addition, the adapter for the Apama Database Connector (ADBC) allows an application to connect to standard ODBC and JDBC data sources as well as Apama Sim data sources. You can add these adapters to an Apama Studio project by the adding the corresponding adapter bundle to the project.

To add an adapter to a project:

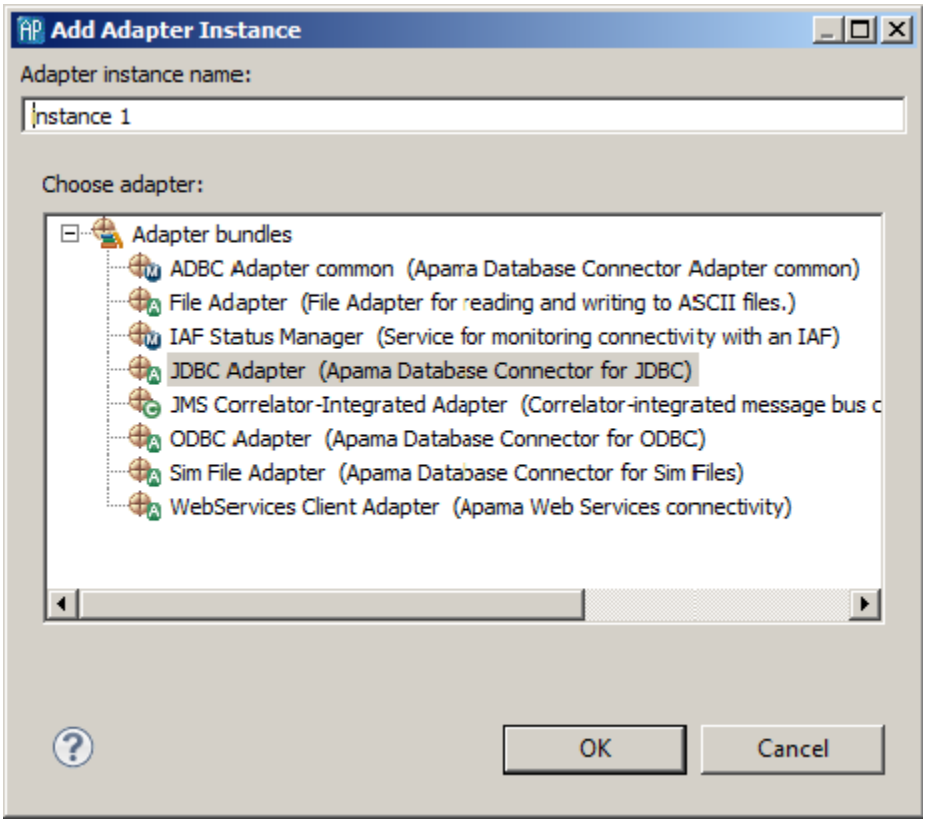
1. There are several ways of adding an adapter to a project.
   - *If you are creating a new Apama project*, select File > Project > New > Apama , give it a name, and click Next. New Apama Project dialog opens.



- *If you are adding an adapter to an existing project*:

a. In the Project Explorer right-click the project and select Apama > Add Adapter. The **Add Adapter Instance** dialog opens.

b. If desired, in the **Add Adapter Instance** dialog, create a new name for the adapter instance or accept the default instance name. Apama Studio prevents you from using a name that is already in use.



2. Select the adapter bundle that is appropriate for your application. Click Finish or OK.

Apama Studio adds an instance of the adapter to the `Adapters` folder in your project along with the supporting monitors such as `IAFStatusManager.mon` and the associated service monitors. Reference to bundled files associated with the adapters, such as IAF Status Manager and Status Support are listed in the adapter's `Dependent Bundles` folder

You can add multiple instances of a given adapter to a project. This allows you create different configurations of the adapter, for example if your application needs to run with different data providers.

For more information on configuring Apama standard adapters, see "Standard Apama Adapters" in *Developing Adapters* (available if you selected Developer during installation) and "Using the Apama Database Connector" in *Deploying and Managing Apama*.

Adding resources to Apama projects

Adding Universal Messaging configuration to projects

Universal Messaging (UM) is Software AG's middleware service that delivers data across different networks. It provides messaging functionality without the use of a web server or modifications to firewall policy. In Apama applications, you can configure and use the connectivity provided by UM.

Apama provides the `UM-config.properties` template file in the `etc` folder of your Apama installation directory. The template is for a standard Java properties file. When you add UM configuration to a project, Apama Studio copies the `UM-config.properties` file to the `config` folder in your project.

To add UM configuration information to a project:

1. Right-click the project you want to add UM configuration properties to.
2. Select Apama > Add UM Configuration.

   A project can contain only one UM configuration properties file. If a `UM-config.properties` file is already in the `config` folder of your project then this option is not available.
3. Right-click the `UM-config.properties` file that is now in the `config` folder of your project. Select Open With > Apama UM Configuration Editor.

   For details about the content of this file, see "Defining UM properties for Apama applications" in *Deploying and Managing Apama Applications*.

For information about using UM in an Apama application, see .

For information about using UM in an Apama application, see "Using Universal Messaging" in *Deploying and Managing Apama Applications*.

Adding resources to Apama projects

Editing Apama files

Apama Studio provides many features to help you write application code for monitor files (`.mon`), block definition files (`.bdf`), event files (`.evt`), and JMon monitor and JMon event files. The following topics provide instructions for using these features:

Working with Projects

Obtaining content assistance

To obtain content assistance, enter Ctrl+space in a blank line, or after one or more words in a line.

Apama Studio displays a list of keywords or names that are valid in that location. For example, it might display a list of event types, standard functions, or actions. Double click the one you want or select with the arrow keys and press Enter.

To ensure that content assistance is always as up-to-date as possible, turn on automatic builds. Automatic builds ensure that when you add, delete, or change a project resource, Apama Studio immediately builds the project. Building a project after a resource change ensures that content assistance has access to the most current resources. See "Build automatically when a resource changes" on page 65.

Using auto-completion

To have Apama Studio automatically complete the word you are typing, enter Ctrl+space.

If you have entered enough characters so that there is only one possible completion, Apama Studio inserts the rest of the word. If there are two or more possible completions, Apama Studio displays a list of the completion candidates. Double-click the correct completion or select with the arrow keys and press Enter.

Apama Studio displays only those completion candidates that are valid in the current context and scope.

To ensure that auto-completion is always as up-to-date as possible, turn on automatic builds. Automatic builds ensure that when you add, delete, or change a project resource, Apama Studio immediately builds the project. Building a project after a resource change ensures that auto-completion has access to the most current resources. See "Build automatically when a resource changes" on page 65.

Displaying information for events and actions

When you position the mouse cursor to hover over an event declaration or action, Apama Studio pops up a box that displays the event type definition or the signature for the action.

Specifying comments

To toggle comment notation, click in a line or select one or more lines, and press Ctrl+/.

Alternatively, you can adjust comment notation as follows:

1. Click in a line or select one or more lines.
2. Right-click anywhere in the code editor.
3. In the menu that appears, select Source and then one of the following:
   - Toggle comment — Inserts `//` comment notation if the selected line or lines is not a comment. Removes `//` comment notation if the selection is already a comment.

- Add block comment — Inserts `/*` at the beginning of the section, and `*/` at the end of the selection.
- Remove block comment — Removes the `/*` and `*/` notation.

Using auto-Indent

To indent one or more lines relative to the entire EPL file:

1. Click in a line or select one or more lines.
2. Press Ctrl+i.

   Alternative: Right-click in the code editor and select Source > Correct Indentation .

Using auto-bracketing

Auto-bracketing is turned on by default.

When auto-bracketing is turned on, and you enter an open bracket ({) or open quotation marks, Apama Studio automatically inserts the closing bracket or closing quotation marks.

To toggle auto-bracketing:

1. In the Apama Studio menu bar, choose Window > Preferences.
2. Expand Apama.
3. Click MonitorScript.
4. Select or clear the Auto Close Brackets checkbox.
5. Click OK.

Using tabs

Apama Studio inserts tabs instead of spaces by default. Each tab is four spaces.

To toggle whether Apama Studio uses tabs or spaces:

1. In the Apama Studio menu bar, choose Window > Preferences.
2. Expand Apama.
3. Expand MonitorScript.
4. Click Editor Formatting.
5. Select or clear the Insert tabs instead of spaces checkbox.
6. If you want, adjust the number of spaces in each tab.
7. Click OK.

Defining shorthand (templates) for frequently used EPL code

EPL templates provide a way for you to define a short name for a longer pattern that you often specify in an EPL file. For example, Apama Studio provides the `dict` template. When you enter `dict` followed by a space, Apama Studio automatically inserts `dictionary<>`.

Apama Studio provides some templates and you can define additional templates. Each Apama project can use all EPL templates.

Apama Studio provides the following EPL templates. To use one, type its name followed by a space.

Name	Description	Pattern
`dict`	`dictionary<>`	`dictionary<%\c>`
`ona`	`on all`	`on all %\c`
`onall`	`on all`	`on al %\c`
`seq`	`sequence<>`	`sequence<%\c>`

In the pattern, `%\c` indicates the location of the cursor after you type the template name.

To define an EPL template:

1. In the Apama Studio menu bar, choose Window > Preferences.
2. Expand Apama.
3. Expand MonitorScript.
4. Click Editor Templates.
5. Click New.
6. In the Name field, specify what you want to enter to insert an instance of the template.
7. In the Description field, specify what Apama Studio should display in the EPL file.
8. In the Pattern field, duplicate what you specified in the Description field and insert `%\c` to indicate the location of the cursor after Apama Studio inserts the value.
9. Click OK twice. All Apama projects can now use this template.

Sharing templates among Apama Studio installations

You can define templates in one Apama Studio installation, and export them for use in another Apama Studio installation.

To export templates:

1. In the source Apama project, define the templates you want to share. See "Defining shorthand (templates) for frequently used EPL code" on page 59.

2. In the **Editor Templates** dialog, click Export All.
3. In the **Export Abbreviations** dialog, specify a text file to contain the templates you are exporting. The default file name is `Abbreviations.txt`. If you can, specify a path that is reachable from Apama Studio installations that will import the templates.
4. Click Save.
5. Click OK.

In some other Apama Studio installation, import the templates as follows:

1. Open the project into which you want to import templates.
2. In the Apama Studio menu bar, choose Window > Preferences .
3. Expand Apama.
4. Expand MonitorScript.
5. Click Editor Templates
6. Click Import.
7. In the **Import Abbreviations** dialog, navigate to a file that contains template definitions.
8. Double-click it.
9. Click OK

Specifying colors to distinguish EPL elements

In EPL files, Apama Studio uses the following colors to indicate the various parts of EPL code:

- Keywords are dark magenta.
- Types are red.
- Comments are green.
- Literal values are blue.
- Operators (+, –, =, and so on) are black.
- All other text is black.

To change one or more colors:

1. In the Apama Studio menu bar, choose Window > Preferences.
2. Expand Apama.
3. Expand MonitorScript.
4. Click Editor Colors.
5. Click the current color of the item whose color you want to change.
6. In the color palette that appears, click the new color.
7. Click OK twice.

Shortcuts when editing Apama files

Apama Studio provides the following shortcuts, in addition to the usual Eclipse shortcuts:

Action	Key	Description
Auto-completion and content assistance	Ctrl+space	If the text you entered has only one possible completion, Apama Studio inserts the completion. If you did not enter any text, or if there are two or more completion candidates, Apama Studio displays candidates for you to choose from.
Toggle comment line notation	Ctrl+/	Toggles comment notation for selected lines. That is, this action inserts or removes `//` from the beginning of each line.
Insert block comment	Ctrl+Shift+/	Makes the selected text a block comment. That is, this action inserts `/*` at the beginning of the selected text, and `*/` at the end of the selected text.
Remove block comment notation	Ctrl+Shift+\	Removes block comment notation from the selected text. That is, this action removes `/*` from the beginning of the selected text, and `*/` from the end of the selected text.
Auto-indent	Ctrl+i	Inserts appropriate indents in selected lines relative to the entire file.
Move line (s) up	Alt+up arrow	Move the selected line or lines to be before the previous line.
Move line(s) down	Alt+down arrow	Move the selected line or lines to be after the subsequent line.
Shorthand for any EPL	Templates	Templates let you define short names for patterns you frequently specify in EPL files. See "Defining shorthand (templates) for frequently used EPL code" on page 59.

Navigating in Apama files

In addition to the usual Eclipse navigation tools, Apama Studio lets you quickly jump to particular parts of an EPL file by

- "Using the Outline view to navigate" on page 62
- "Using the Quick Outline to navigate" on page 62
- "Jumping to an event or action definition or variable declaration" on page 62"Adding JMon classes" on page 47
- "Searching in EPL files" on page 62

You can use these features to navigate the code of EPL files (`.mon`), and block definition files (`.bdf`).

Working with Projects

Using the Outline view to navigate

In an Apama Studio perspective, the Outline view displays the event types and monitors in the currently open `.mon` or `.bdf` file, showing event parameters and actions and monitor fields, actions, and listeners within actions. When you click an entry in the Outline view, the focus in the editor view jumps to the code that defines the item you clicked.

Apama Studio updates the Outline view each time it saves your EPL file. If you add a resource to your EPL file but you do not save your file, the new resource does not appear in the Outline view until you save your file.

Using the Quick Outline to navigate

While you are viewing a `.mon` or `.bdf` file in the Apama Studio editor, you can use the Quick Outline to jump to an event type, monitor, or action. Press Ctrl+O or select Navigate > Quick Outline to display the Quick Outline. Apama Studio pops up a list of event parameters and actions and monitor fields, actions, and listeners within actions for the current EPL file. Click an entry in the list to jump to the code that defines the item you click.

Jumping to an event or action definition or variable declaration

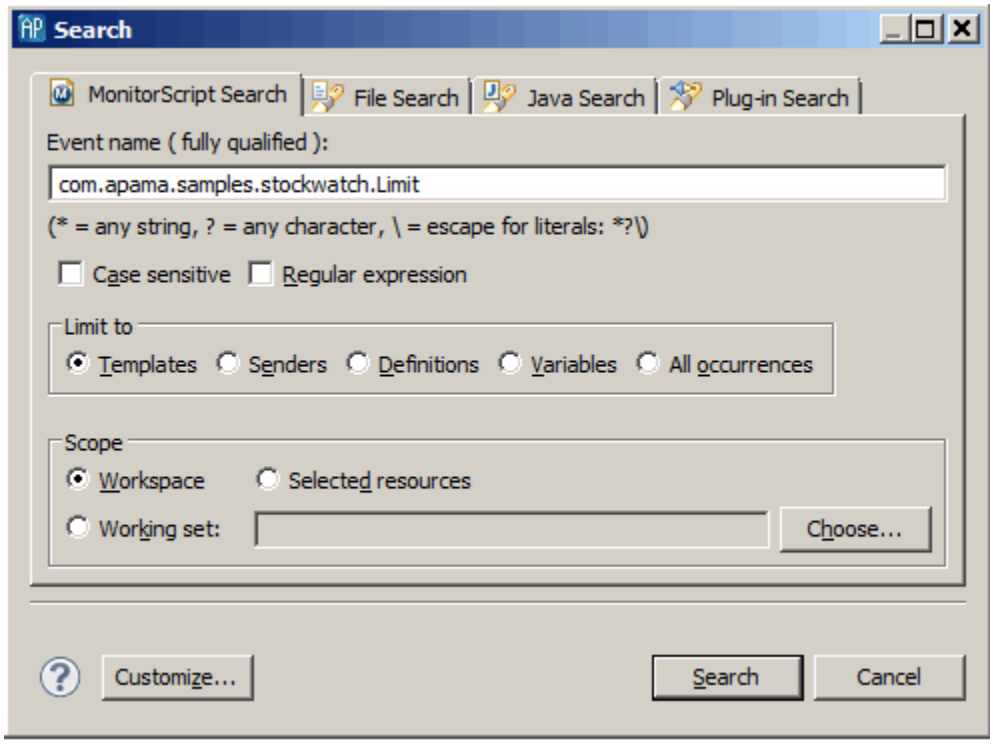
To jump to an event or action definition or variable declaration:

1. Select a reference to the event, action, or variable whose declaration you want to view.
2. Press F3 or choose Navigate > Open Declaration from the Apama Studio menu bar. Apama Studio highlights the line that begins the declaration for the selected event or variable.

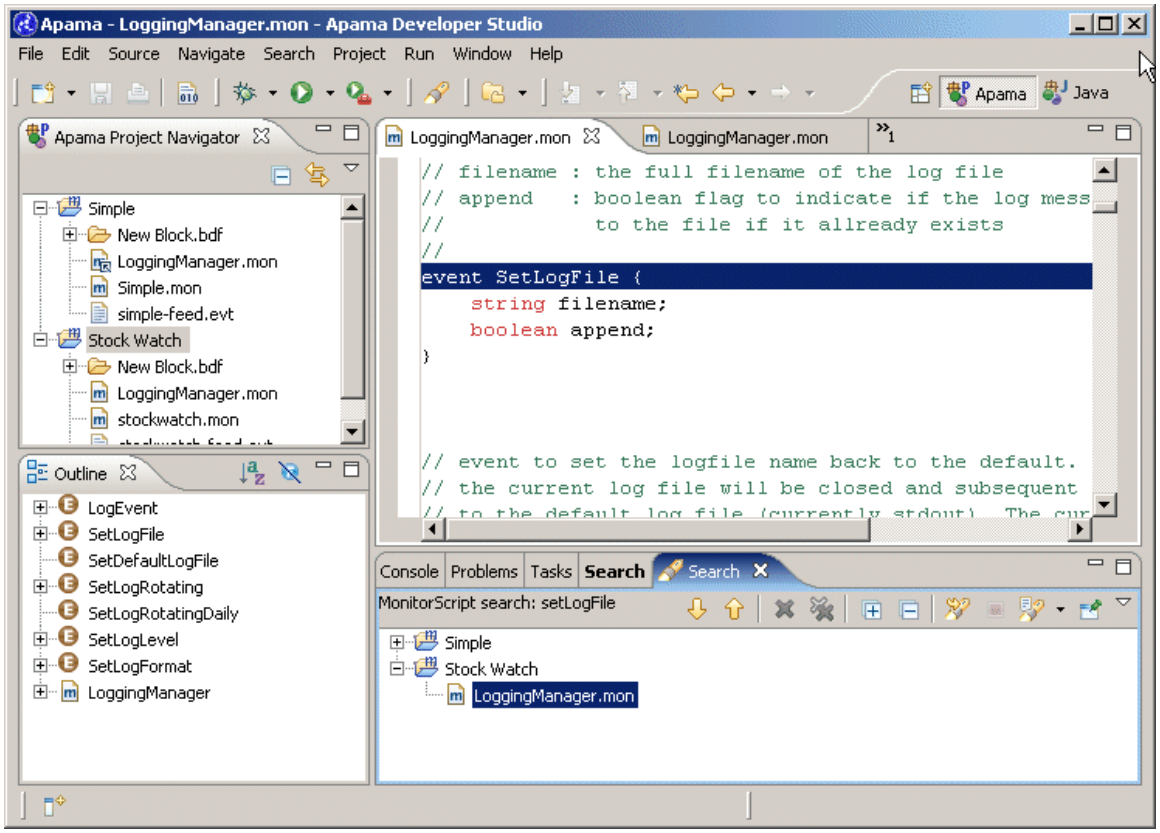
Searching in EPL files

You can search for event types in your project's `.mon` or `.bdf` files. To start a search:

1. In either the **Developer** or **Workbench** perspective, select Search > Search from the menu bar. In the **Developer** perspective, you can also click the Search icon  . The **Search** dialog is displayed. Make sure the MonitorScript Search tab is selected.



2. Type the event type you want to search for in the Event name (fully qualified) field. You can use wildcard characters and regular expressions in your search. Modify the search details as necessary, for example by specifying if the search should be case sensitive or if you want to limit the scope of the search. You can limit your search to templates (event listeners), senders (route, enqueue, and emit), and variables (event types are used as variables).
3. Click Search. Results are displayed in the Search view (Search tab), showing the files where the search term occurs and how many times it occurs. To display the first occurrence of the search term, double click the file name or click the Show Next Match button. This opens the file containing the term in the appropriate editor. You can navigate through all occurrences of the search term with Show Next Match and Show Previous Match buttons.



Building Apama projects

In Apama Studio, “building a project” refers to the process in which the project is validated, and any errors or warnings flagged to the user. This section describes the features Apama Studio provides to control when Apama Studio will build an Apama project.

Apama Studio includes options to:

- "Build automatically when a resource changes" on page 65
- "Build all Apama projects" on page 65
- "Build one Apama project" on page 65
- "Build a working set" on page 65
- "Clean and rebuild projects" on page 66

Apama Studio tries to complete each build. It does not stop when it finds an error. If Apama Studio finds problems during validation, it indicates them as follows:

- In the Problems view, the error icon appears at the beginning of each line that describes an error.
- In the Project view, the error icon appears at the beginning of the name of a project that contains at least one file that is not valid.
- In the Project view, the error icon also appears at the beginning of the name of each file that is not valid.

- In the file editor, the error icon appears at the beginning of each line that contains an error.
- In the overview scroll bar to the right of the editing pane, regardless of how long the file is, a mark appears for every error in the file. You can hover over a mark to view a description of the error that it flags, or you can click on any mark to display the line that contains that error.

Working with Projects

Build automatically when a resource changes

Apama Studio can automatically build a project whenever you add, delete, or change a resource in that project. This is the default and Apama strongly recommends that you leave this feature on. Building a project automatically ensures that content assistance menus are always up to date.

To toggle Build Automatically on or off:

1. Select Project in the Apama Studio menu.
2. In the drop-down menu that appears, if a checkmark appears in front of Build Automatically, automatic builds are already turned on. Otherwise, select Build Automatically to turn it on.

Build all Apama projects

To validate all Apama projects in your workspace, choose Project > Build All from the Apama Studio menu bar.

Build one Apama project

To validate one Apama project:

1. In the Apama Project Explorer view, select the project you want to build.
2. In the Apama Studio menu bar, choose Project > Build Project.

Build a working set

When an Apama project is large, you might find it more efficient to build a subset of the project rather than the entire project. To build a subset of an Apama project, specify a working set that contains the files you want to build. Then build the working set you defined.

To define a working set:

1. In the Apama Studio menu bar, choose Project > Build Working Set > Select Working Set.
2. In the **Select Working Set** dialog, click New.
3. Double-click Resource.

4. Specify a name for this working set.
5. Expand the project(s) that contains the file(s) that you want to be in the working set.
6. Select the file(s) that you want to be in the working set, and click Finish.
7. Back in the **Select Working Set** dialog, select the working set you just defined and click OK.

To build this working set, choose Project > Build Working Set > working_set_name from the Apama Studio menu bar.

Clean and rebuild projects

When Apama Studio cleans a project, it discards all build problems and build states. To rebuild a project from scratch, as though you have never built it before:

1. In the Apama Studio menu bar, choose Project > Clean.
2. Indicate which projects you want to rebuild from scratch or select Clean all projects.
3. Click OK.

Configuring the project build path

By default, Apama Studio will include all of the files in a project when building (and launching) it. However, for many applications it will be necessary to customize the build path, adding additional files from outside the project (and less commonly, limiting the set of files under the project directory that will be included). The set of files under the project directory that will be build can be customized using the Source tab. Apama Studio provides three ways to add additional files from outside the project to its build path:

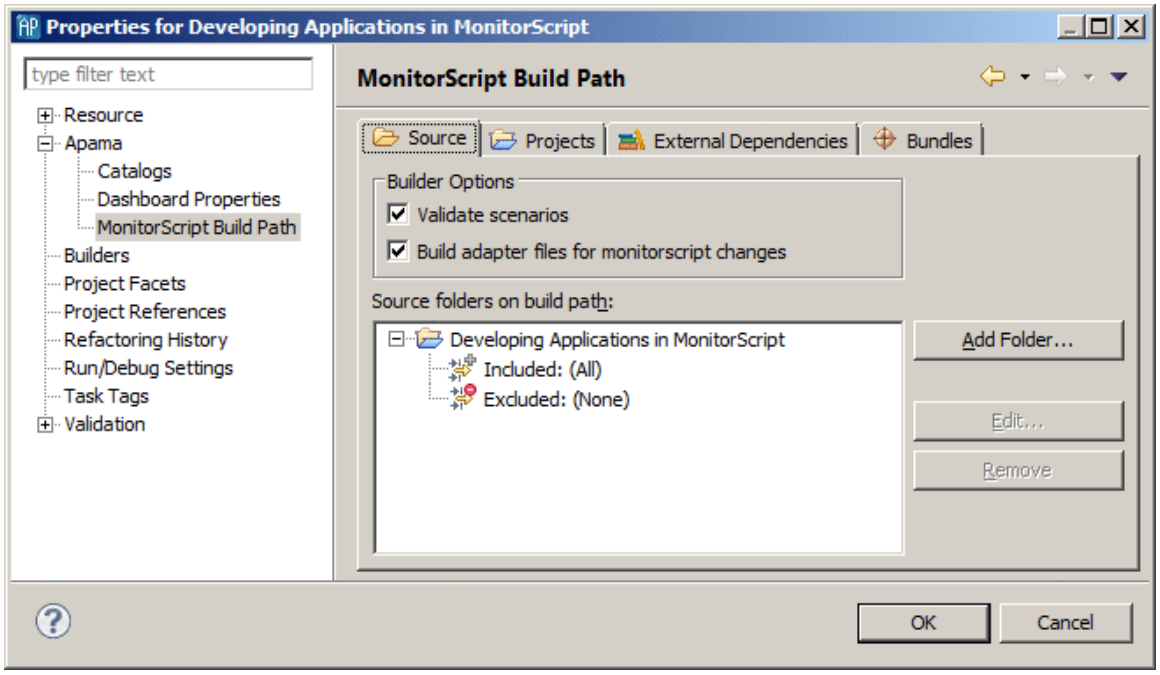
- Required Projects
- External MonitorScript Dependencies
- Bundles

Project source files

By default, all files in the project directory are included in the build path, but this can be customized. Files within a project are grouped into folders and you can specify which folders should be included when a project is built. In addition, within each folder you can specify patterns that determine which files should be included or excluded.

To specify files this way:

1. Select the project in the Project view and select Project > Properties from the Apama Studio menu.
2. Expand Apama.
3. In the **Properties** dialog, select MonitorScript Build Path, and display the Source tab.

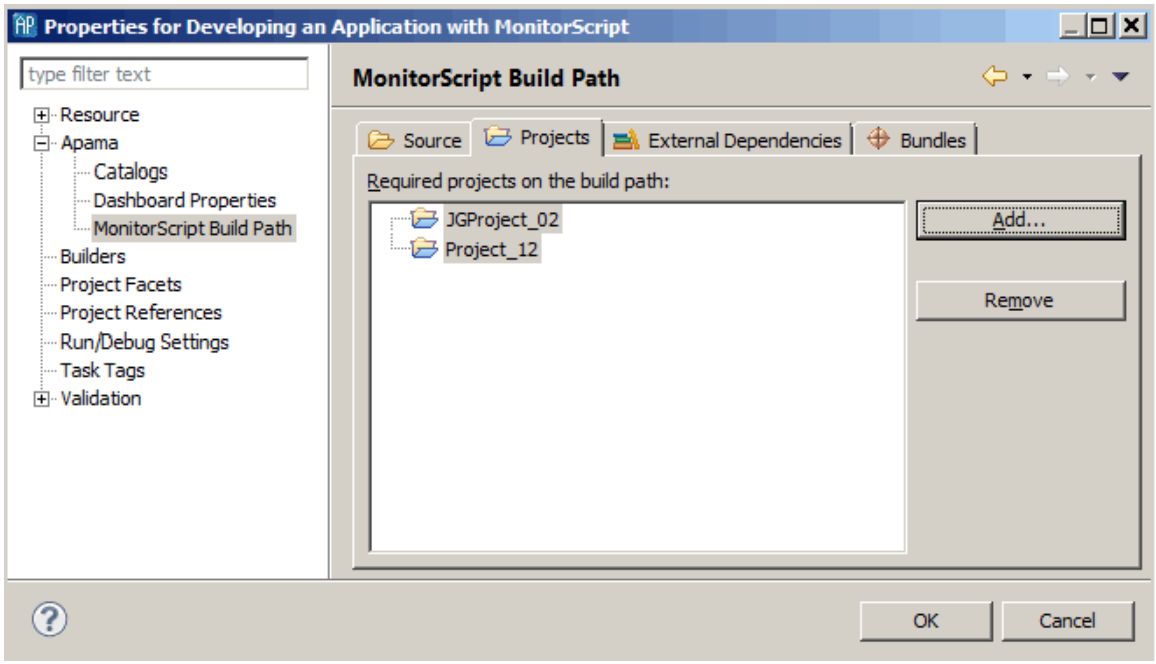


4. On the Source tab you add new folders and specify which files to include in the build process. For example, you can include all event files with the `.evt` extension. For folders in the project, you can also modify which files to include or exclude.

Specifying projects

A project can make use of other Apama Studio projects. To specify that your project should include another project in the build process:

1. Select the project that requires another project in the Project view and select Project > Properties from the Apama Studio menu.
2. Expand Apama.
3. In the **Properties** dialog, select MonitorScript Build Path, and display the Projects tab.



4. On the Projects tab, click Add to display the **Required Project Selection** dialog where you specify which projects to include in the build process.

Specifying external dependencies

When an EPL file in your project depends on an external EPL file, you must explicitly specify the dependency. For example, if you refer to event types or actions that are defined in other EPL files, you must specify the files that define those constructs. This ensures that Apama Studio includes the external file in the validated project. It also ensures that the Apama Studio builder, content assistance facility, and launcher have access to the most up-to-date version of each required file.

How you define dependencies depends on whether you are the only one using your project or you share the project with one or more users.

Specifying dependencies for a single-user project

When you are the only one using a project, specify a file that your EPL file depends on as follows:

1. Select the project for which you want to define a dependency.
2. In the Apama Studio menu bar, choose Project > Properties.
3. In the **Properties** dialog that appears, select Apama.
4. Click MonitorScriptBuild Path.
5. Display the External Dependencies tab.
6. Click Add External.
7. In the **MonitorScript File Selection** dialog, navigate to the file your project requires.
8. Double-click the file.
9. Click OK in the **Properties** dialog to specify the dependency.

Specifying dependencies for a multi-user project

When Apama Studio builds a project, it defines any dependencies in an XML file that hardcodes the path to the external file(s). This works correctly if you do not share projects among multiple users. However, if two or more users share a project, they might not store external files in the same location.

If you are sharing projects among two or more users, you should use a variable to define an external file on which your EPL file depends. For example, Apama Studio automatically defines `APAMA_HOME` and `APAMA_WORK` variables that can be used to locate files under either of those directories in a way that is independent of exactly where Apama has been installed on the current machine. To use a variable to define a dependency:

1. Select the project, or a file in the project, for which you want to define a dependency.
2. In the Apama Studio menu, choose Project > Properties.
3. In the **Properties** dialog that appears, expand Apama.
4. Click MonitorScriptBuild Path.
5. Display the External Dependencies tab.
6. Click Add Variable.
7. In the **New Variable Dependency Entry** dialog that appears, do one of the following:
   - If a variable already exists that identifies the required file, double-click that variable. Then click OK. You are done and can skip the remaining steps.
   - To define a new variable that identifies the required file, click Configure Variables.
8. In the **New Variable Entry** dialog, in the Name field, enter the name of the new variable. The convention is to use `UPPER_CASE` names for build path variables (for example, `APAMA_HOME`).
9. Click File... or Folder... according to which one you want the variable to represent.
10. Navigate to the appropriate folder or EPL file and double-click it.
11. Click OK twice.
12. Click OK in the **Preferences** dialog; Apama Studio confirms that the classpath variables have changed, and prompts you to indicate whether you want to rebuild your project so that it can use the new variable. Click Yes.

Defining MonitorScript Build Path variables

In any Apama project, you can define a variable that you can use in all your Apama projects. You might find it useful to define a variable in a situation where multiple users share the same project. If the project is dependent on one or more external files, not all users might have the external file stored in the same location. Each user defines a variable to specify the location of a shared file dependency.

To define an Apama project variable:

1. In the Apama Studio menu bar, choose Window > Preferences.
2. In the **Preferences** dialog that appears, expand Apama.
3. Click MonitorScript Path Variables.

4. In the **MonitorScript Path Variables** dialog, click New.
5. In the **New Variable Entry** dialog, in the Name field, enter the name of the new variable.
6. Click File... or Folder... according to which one you want the variable to represent. Choose whatever is convenient for you. If the variable represents a directory, you can use the same variable in the path for more than one file. If the variable represents a file, you can use that variable for only that file's path.
7. Navigate to the appropriate folder or EPL file and double-click it.
8. Click OK and in the **New Variable Entry** dialog, click OK again.
9. Click OK in the **Preferences** dialog; Apama Studio displays the **Build path Variables Changed** dialog asking if you want to perform a full build of your project. Click Yes to ensure that the current project state is consistent with the new variable. If you are sure that the new variable does not affect the current project state, you can click No.

Importing projects

To import an Apama Studio project created on another machine into this Apama Studio workspace:

1. Select File > Import > General > Existing Projects into Workspace from the Apama Studio menu. Click Next.
2. In Select root directory specify the root directory where the project is located.
3. Add a check box next to the project(s) you want to import.
4. Check the Copy projects into workspace checkbox if you want to copy all the project files into the workspace directory (located under `APAMA_WORK`)

   Leave the check box unchecked if you want to simply link to the project in its current location instead.

If you have an Apama application that is not currently part of an Apama project:

1. Select File > New > Apama Project from the Apama Studio menu.
2. Enter the name of the new project.
3. Uncheck the Use default location check box.
4. Specify the folder containing the files that you wish to import as a new project.
5. Click Finish.

A project that you import might have dependencies on environment variables, bundles, blocks, or functions that have not yet been added to Apama Studio. As an alternative to explicitly adding each dependency, see "Setting up the environment before importing projects" on page 78.

Working with Projects

Importing adapter configurations

You can import a Web service client adapter configuration from an archive file that has been generated in Apama Studio. To import a Web service client adapter configuration into an Apama Studio project:

1. In the **Project Explorer** view, right-click the name of the project and select Import from the pop-up menu. This displays the **Import** dialog.
2. In the **Import** dialog, expand Apama.
3. Select Adapter Configurations and click Next. The **Adapter Configurations Import Wizard** appears.
4. In the **Adapter Configurations Import Wizard**, specify the archive file that contains the adapter configuration you want to import. You can use the Browse button to navigate to the desired project.
5. Accept or specify the name of the project into which you want to import the adapter configuration. You can use the Browse button to navigate to the desired location.
6. Click Finish. This adds the adapter configuration to the specified Apama project.

Working with Projects

Exporting project information

You can export a project's information for the following purposes:

- "Exporting a project initialization file list" on page 71
- "Exporting to a deployment script" on page 72
- "Exporting scenarios" on page 75
- "Exporting Correlator Deployment Packages" on page 75
- "Exporting adapter configurations" on page 75
- "Exporting ApamaDoc" on page 76

Working with Projects

Exporting a project initialization file list

To export a project initialization file list:

1. Select File > Export from the Apama Studio menu.
2. Expand Apama.
3. Select Project Initialization File List and click Next.
4. If you want the file list to also contain all event files from the project, select Include event files (*.evt).
5. Specify a name for the file to hold the initialization file list and click Finish.

This creates an ordered list of the files on the project build path and saves it in a text file (with one entry per line). This file can be imported into an Apama component's Initialization tab in the

Enterprise Management and Monitoring console (EMM) to help convert a development Apama Studio project into a production EMM deployment.

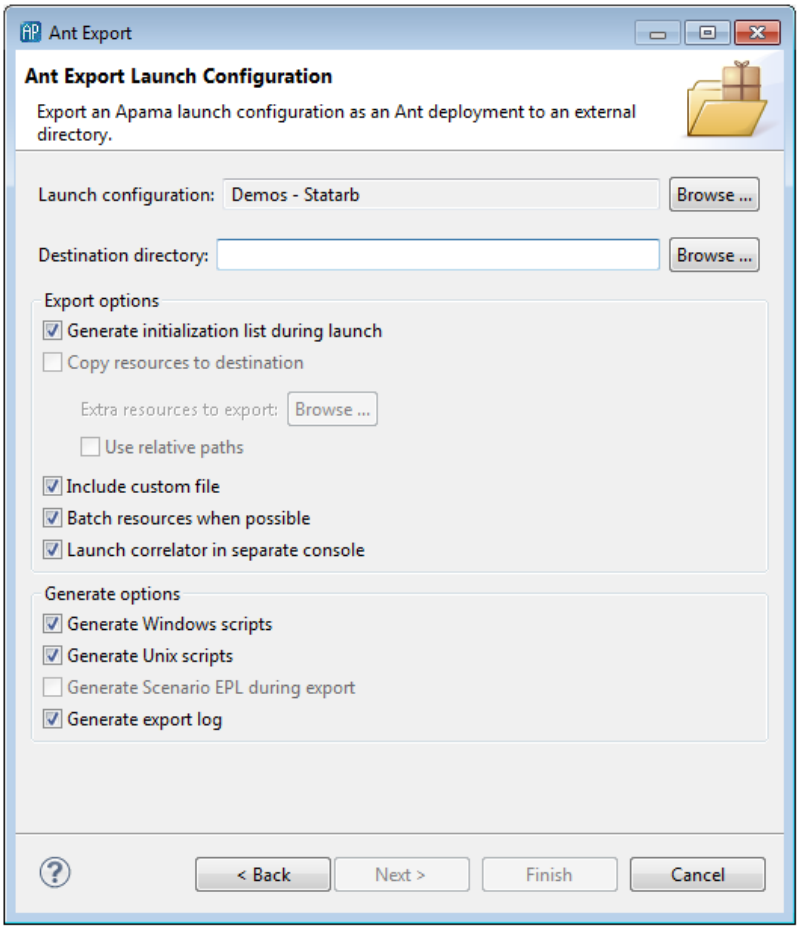
Note that at present this list does not include any scenario (`.sdf`) files on the build path. Scenarios should therefore be exported to an EPL file manually (using Event Modeler), and the EPL files added directly to EMM's initialization list once the Apama Studio Initialization File has been imported.

Exporting to a deployment script

You can export an application's launch configuration to create a deployment script. This generates the build files, configuration files, property definition files, scripts, and EPL files from scenarios and copies other resources such as dashboards that are used by Ant to build and launch the project on a different machine.

To export an Apama launch configuration to a deployment script:

1. In the Project Explorer view, right-click the name of the project and select Export from the pop-up menu. This displays the **Export** dialog.
2. In the **Export** dialog, expand Apama.
3. Select Apama Ant Export and click Next. The **Ant Export** dialog appears.



4. In the **Ant Export** dialog, specify the settings as follows:

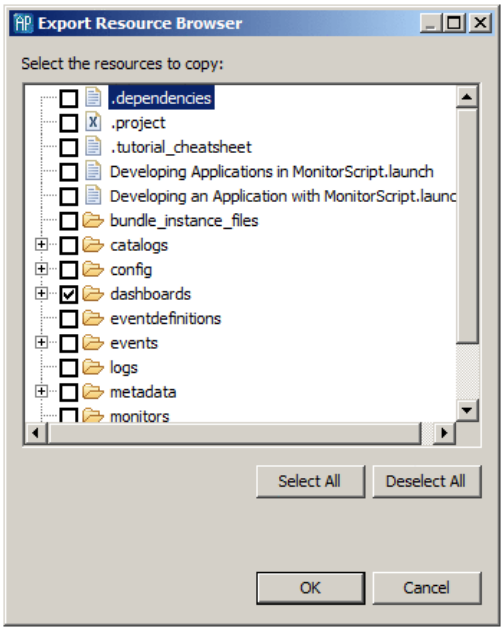
- Launch configuration — The export operation uses the project's default launch configuration, but if a project has multiple launch configurations, you can select which one to export. If you want to select a different launch configuration, click Browse. This will display the **Choose Launch Configuration** dialog.
- Destination directory — The name of the directory for the exported files.
- Generate initialization list during launch — Dynamically creates the file injection list from the project directory when the exported deployment script is executed, rather than during the export process.

  When you select this option you need to export your project's launch configuration only once. The generated scripts specify the location of your project directory and then use the content of your project directory to create the file injection list at deployment time. If you do not select this option, then you must re-export the configuration each time you add, remove or edit a file in your application.

  For JMon applications that you develop in Apama Studio, Apama Studio creates the required `.jar` files whenever you modify your Java files. If you do not develop your JMon application in Apama Studio, see *Generating deployment descriptor files from annotations* in *Developing Apama Applications in Java* for information about building the `.jar` file for your application. Ensure that your application's `.jar` file is in your project directory before export.

  If you selected a shared location when you created the launch configuration that you are exporting then Apama Studio generated two files that contain the launch information (`.deploy` and `.launch`) and put them in the specified shared location. After you create the launch configuration, any changes you make to the launch configuration are reflected in these files. Since the exported deployment script uses these files at deployment time, any launch configuration changes will also be reflected upon deployment. Except, if you change the shared location then you must re-export the launch configuration to a new Ant deployment script. If you do not, the old Ant deployment script fails because it cannot find the `.deploy` file.

  When Generate initialization list during launch is selected the option to Generate Scenario EPL during export is not available. Instead, EPL `.mon` files are always created from scenario files upon deployment. Also, Copy resources to destination is not available because the script points to the project directory.
- Copy resources to destination — By default, the export operation copies only the project's dashboard definitions. To change the specific resources that Apama Studio exports, click Browse to display the **Export Resource Browser** dialog and specify the resources you want.



- Use relative paths — By default, the generated `build.xml` file uses relative pathnames for the application's monitors, events, scenarios, jars, and adapter configurations and properties. Uncheck this box if you want to use absolute pathnames.
- Include custom file — If you want the exported launch configuration to perform other operations, select this option to generate a stub `custom.xml` file. The `custom.xml` file has `pre-custom` and `post-custom` targets where you can add the desired operations.
- Batch resources when possible — By default, the `build.xml` file generated by the export operation specifies that all monitors in the project will be injected in a batch when the application is launched. If you want to inject each monitor separately when the application is launched, uncheck this check box.
- Launch correlator in separate console — By default, the exported launch configuration will launch the correlator in a separate console. Uncheck this check box if you want to start the correlator in the console where the launch is started.
- Generate Windows scripts — Exports all scripts used by the launch configuration in Windows form.
- Generate Unix scripts — Exports all scripts used by the launch configuration in UNIX form.
- Generate Scenario EPL during export — Generates EPL files for scenarios during the export operation. If you do not select Generate initialization list during launch and you want to copy the scenario definition files instead of generating EPL files, uncheck this check box. Note that when you select Generate initialization list during launch any scenarios are converted to EPL `.mon` files when the launch configuration injects the files into the correlator. Consequently, this option is not available when Generate initialization list during launch is selected.
- Generate export log — By default the export operation generates a log file that records the export operation. Uncheck the check box if you do not want to record the log file.

Click Finish. Apama Studio generates the files in the specified destination directory.

Exporting scenarios

From Apama Studio's **Export** dialog, you can export a project's scenarios in two ways:

- As EPL code — See "Exporting scenarios as EPL" in *Developing Apama Applications in Event Modeler.*
- As block templates — See "Exporting scenarios as block templates" in *Developing Apama Applications in Event Modeler.*

Exporting Correlator Deployment Packages

You can export a project's EPL and scenario files to a Correlator Deployment Package (CDP). CDP files use a proprietary, non-plaintext format that treats EPL and scenario files in a manner similar to the way a JAR file treats a collection of Java files.

1. In the **Project Explorer** view, right-click the name of the project and select Export from the pop-up menu. This displays the **Export** dialog.
2. In the **Export** dialog, expand Apama.
3. Select Export as Correlator Deployment Package and click Next. The **Correlator Deployment Package** wizard is displayed.
4. On the first page of the **Correlator Deployment Package** wizard, if necessary specify a project in the Project field. You can use the Browse button to navigate to the desired project.
5. Also on the first page of the **Correlator Deployment Package** wizard, in the Package Filename field, specify the name of the CDP file you want to create. You can use the Browse button to navigate to the desired location.
6. Click Next. The second page of the **Correlator Deployment Package** wizard is displayed.
7. On the second page of the **Correlator Deployment Package** wizard, specify the files you want to add to the package as follows:
   a. On the Artifacts Selection tab, select the EPL and scenario files you want to include in the package.
   b. On the Injection Order tab, use the Move Up and Move Down buttons to specify the order in which you want to inject the EPL and scenario files.
8. Click Finish.

Exporting project information

Exporting adapter configurations

You can export a project's Web service client adapter configuration to an archive file, which you can generate in Apama Studio. To export a Web service client adapter configuration from an Apama Studio project:

1. In the **Project Explorer** view, right-click the name of the project and select Export from the pop-up menu. This displays the **Export** dialog.
2. In the **Export** dialog, expand Apama.
3. Select Adapter Configurations and click Next. The **Adapter Configurations Export Wizard** appears.
4. In the **Adapter Configurations Export Wizard**, if necessary, specify a project in the Project field. You can use the Browse button to navigate to the desired project.
5. In the Adapter Configurations field, select the adapter configurations you want to export.
6. In the Adapter Resources field, select the adapter resources you want to export.
7. In the To archive file: field, specify the archive file that you want to contain the exported adapter configuration. You can use the Browse button to navigate to the desired location.
8. Click Finish. This creates an archive file in the specified location. You can import this archive into any Apama project.

Exporting project information

Exporting ApamaDoc

In Apama Studio you can use the ApamaDoc tool to generate reference documentation for the EPL source code you add to a project. From the **Export** dialog, select ApamaDoc Export. This generates static HTML pages that document the structure of all EPL code in a project.

For detailed information on how to annotate your EPL source code and generate ApamaDoc, see "Generating documentation for your EPL code" in *Developing Apama Applications in EPL.*

Deleting projects and resources

If you want to delete projects or resources you should use Apama Studio to do so, rather trying to delete them directly from the file system.

Working with Projects

Deleting resources

To delete a resource from a project:

1. In the **Project Explorer** view or **Workbench Project** view, right-click the resource and select Delete from the pop-up menu or select the resource and select Edit > Delete from the Apama Studio menu. In the **Workbench Project** view you can select the resource and click the Delete button.
2. In the **Confirm Resource Delete** dialog, click Yes if you want to proceed.

Deleting projects and resources

Deleting projects

To delete a project:

1. In the **Project Explorer** view or **Workbench Project** view, right-click the project and select Delete from the pop-up menu or select the resource and select Edit > Delete from the Apama Studio menu. In the **Workbench Project** view you can select the resource and click the Delete button.
2. In the **Confirm Project Delete** dialog select whether or not you want to delete all the project's resources from the filesystem, or simply remove the project from the Project view leaving all files in place. After selecting the latter option, such projects can be added to the workspace again using the **Import** wizard (see "Importing projects" on page 70). In most cases it is more useful to select the option to delete the project's contents at the same time as the project itself, to avoid confusion.

Deleting projects and resources

Adding the Apama nature to a project

If you are working with a project that is not an Apama project, for example, a Java project, you can apply the Apama nature to the project with Apama Studio. The project then becomes an Apama project in addition to whatever natures it had before.

To add the Apama nature to a project:

1. In the Project Explorer view (or in the Package Explorer view if the project is a Java project), right click the project.
2. Select Apama > Add Apama Nature from the pop-up menu.

After you apply the Apama nature to a project, the project shares all the features of any other Apama project, for example, EPL errors will be detected and flagged, and the project can be launched in an Apama correlator.

Note that the Add Apama Nature menu item is not available if a project is already an Apama project.

Working with Projects

Internationalizing Apama applications

By default, Apama Studio saves Apama project files in your platform's native encoding. If you choose to save Apama project files in UTF-8 encoding in the Resource tab of a file or a folder's **Properties** dialog, Apama Studio adds a UTF-8 BOM character at the beginning of each file. This indicates that the contents are in UTF-8. The character is required to be compatible with other Apama tools.

To specify the encoding for Apama projects:

1. In the Apama Project Navigator view, click the project for which you want to define the encoding.

2. In the Apama Studio menu bar, choose Project > Properties.
3. In the **Properties** dialog, click Info.
4. In the Text File Encoding group box, click Other and select the encoding you want.
5. Click OK.

Working with Projects

Checking the error log

While using Apama Studio, you might receive a message that prompts you to check the error log. If the Error Log tab does not already appear with the Problems, Tasks, and Console tabs, display it as follows:

1. In the Apama Studio menu, choose Window > Show View > Other.
2. In the **Show View** dialog, expand General.
3. Double-click Error Log. The Error Log tab now appears with the Problems, Tasks, and Console tabs.

Please see the *APAMA_WORK*\logs\apama_studio.log file for more detailed information about any unexpected problems you encounter while using Apama Studio

Working with Projects

Setting up the environment before importing projects

A project that you want to import into Apama Studio might have dependencies on any of the following:

- Environment variables
- Catalogs of blocks, bundles or functions
- String substitutions

Before you can build your project, you would need to add each dependency to Apama Studio. An alternative to adding each dependency is to define the dependencies in a file, and place the file in the $APAMA_HOME\studio\extensions folder. When Apama Studio starts it collects any files in its extensions folder and uses them to set up the Apama Studio environment. When you then import your project its dependencies will already be in place.

Format of extensions file

A file in the extensions folder must have the .ste (Studio Tuning Extension) extension and the data it contains must be in the following format:

- Define each item to be added to Apama Studio on its own line.
- The first value in each line must be the type of the item you want to add. The type must be one of the following:
  - BLOCK_CATALOG

  - BUNDLE_CATALOG
  - FUNCTION_CATALOG
  - STRING_SUBSTITUTION
  - VARIABLE
- In each line, insert a semicolon between values.
- Insert # at the beginning of a line to make it a comment.
- The values you specify vary according to the specified type. Path specifications must be fully qualified; they cannot be relative. In a path specification, you can use a variable, which can be defined directly in Apama Studio or in any .ste file. You can specify the items in any order.
  - For block, bundle, or function catalogs, the format is the type followed by a path. For example:

    ```
    BLOCK_CATALOG ; C:\Program Files\MyCompany\MyApp\MyBlockCatalog
    ```
  - For string substitutions, the substitution value cannot be edited or removed in Apama Studio. For details, See Java development user guide > Reference > Preferences > Run/Debug > String Substitutions in the Eclipse help provided with Apama Studio. The format is as follows:

    ```
    STRING_SUBSTITUTION ; variable_name ; path ; description
    ```

    For example:

    ```
    STRING_SUBSTITUTION ; HOME ; C:\MyApp ; Install dir for my app
    ```
  - For variables, the variable's value cannot be edited and the variable cannot be removed in Apama Studio. The format is as follows:

    ```
    VARIABLE ; variable_name ; path
    ```

    For example:

    ```
    VARIABLE ; HOME ; C:\MyApp
    ```

Suppose you have a .ste file in place and you start Apama Studio. If you subsequently modify the content of that .ste file you must restart Apama Studio for the changes to take effect.

Results of using an extensions file

After you define an extensions file, place it in the $APAMA_HOME\studio\extensions folder and then start Apama Studio.

You should see the items you defined in the extensions file in the appropriate Apama Studio dialogs. For example, select Window > Preferences and then expand Apama and click Catalogs. Any catalogs you specified in the extensions file should appear in the appropriate tab. However, you cannot edit or remove any catalogs that were added to Apama Studio by means of an extensions file.

If you import a project that uses any of the items specified in the extensions file then the imported project will be valid with regard to any of these dependencies.

Working with Projects

Using Apama Studio to configure adapters that use UM

You can use Apama Studio to configure an IAF adapter to use Universal Messaging. To do this:

1. Add support for UM to your project. See "Adding Universal Messaging configuration to projects" on page 55.
2. Add one of the following adapters to your project:
   - File Adapter
   - JDBC Adapter
   - ODBC Adapter
   - Sim File Adapter
   - WebServices Client Adapter
3. Open the instance of the adapter you just added. In the Settings tab, you can see the following substitution variables:
   - `APAMA_MSG_ENABLED`
   - `UM_MSG_ENABLED`

   Use Apama Studio's launch configuration editor to manage these substitution variables.
4. Select Run > Run Configurations....
5. In the Run Configurations dialog, select the Apama project that contains the adapter you want to configure and then click the Components tab.
6. In the Components tab, double-click the adapter you want to configure.

   In the Adapter Configuration dialog, if you added UM support to your project, the default for a new adapter in your project is that the UM Messaging radio button is selected. Adding UM support to a project does not change an existing adapter configuration.
7. Ensure that UM Messaging is selected. This sets the related substitution variables as follows:
   - `APAMA_MSG_ENABLED` is false.
   - `UM_MSG_ENABLED` is true.
8. Click OK , Apply, and then Close to save the adapter configuration in that launch configuration.

Apama Studio manages the `APAMA_MSG_ENABLED` and `UM_MSG_ENABLED` substitution variables by means of the launch configuration.

If you click the XML Source tab you can see that the `<apama>` element and the `<universal-messaging>` element each contain the `enabled` attribute. In the `<apama>` element, Apama Studio sets the `enabled` attribute to the `@APAMA_MSG_ENABLED@` substitution value. In the `<universal-messaging>` element, Apama Studio sets the `enabled` attribute to the `@UM_MSG_ENABLED@` substitution value. When you launch the project Apama Studio uses the settings of the `APAMA_MSG_ENABLED` and `UM_MSG_ENABLED` substitution variables to set the value of the `enabled` attribute in each element.

The default is that the `enabled` attribute is set to true. Consequently, if you delete the `enabled` attribute, it is as if it is set to true.

See also: "Configuring adapters to use UM" in *Deploying and Managing Apama Applications*.

Working with Projects

Chapter 3: Creating Blocks

Apama comes with many standard blocks that you can use in your scenarios. In addition, you can use Apama Studio to create your own blocks to implement specialized behavior. This section describes how to create custom blocks for use by scenarios in your Apama applications.

About blocks

Blocks are modules that you can import and use within your scenarios in Apama's Event Modeler. Blocks accept inputs, execute logic of their own, and generate output. Their primary purpose is to provide scenarios with access to complex functionality that can only be programmed in Apama Event Processing Language (EPL). They also provide an element of reuse. EPL is the new name for MonitorScript and is the native language of the event correlator.

For more information on writing EPL code, see "Getting Started with Apama EPL" in *Developing Apama Applications in EPL*.

Apama is distributed with a library of blocks that perform a variety of tasks such as general and financial analysis, order management, and timing. For more information on these, see "Using Standard Blocks" in *Developing Apama Scenarios*. If an application requires additional functionality, developers can create their own custom blocks.

The following topics provide more introductory information about blocks:

- "Introduction to block definition files" on page 81
- "Description of block interface elements" on page 82
- "How scenarios communicate with their blocks" on page 82

Creating Blocks

Introduction to block definition files

A block is defined in a block definition file, which has a `.bdf` extension. This XML file describes the functionality of the block and includes its implementation in EPL. With Apama Studio, you graphically define the block's interface, and Apama Studio automatically generates all the XML elements of the `.bdf` file. In addition, Apama Studio generates skeleton EPL code for the block's behavior. Apama Studio provides a dedicated editor where you add your custom code and it validates the EPL code you add.

The block definition file actually defines an *EPL template*. The term "template" is used because the EPL in the `.bdf` is not complete EPL code. Instead of the actual block name, the `.bdf` code uses a

specially encoded stand-in for the real block name. The real names are automatically generated when the combined scenario and block are converted into full EPL code when they are injected into the correlator.

Description of block interface elements

A block's interface consists of the set of parameters, input feeds, output feeds and operations it defines. You specify these items when you create a block with Apama Studio. Apama Studio then generates the corresponding actions.

- **Parameters** — Parameters configure the behavior of a block. You typically use parameters to initialize the block or to modify its core behavior.
- **Input feeds** — Input feeds connect live data streams to blocks. In each block input feed, you define input fields and map data in the stream to the appropriate input field. All the fields of an input feed are updated simultaneously.
- **Output feeds** — Output feeds stream output data generated by the block. Each output feed is a collection of fields that all get updated simultaneously.
- **Operations** — Operations are specific behaviors that the scenario invokes, such as starting or stopping the processing of data.

For examples, see "Using Standard Blocks" in *Developing Apama Scenarios*.

How scenarios communicate with their blocks

Apama Studio implements a block as an event type. When you create a block, Apama Studio generates the event type definition for that block. The block's event type definition includes a number of actions that Apama Studio defines for you and that you can edit.

Communication from a scenario to a block instance is accomplished through calls to these actions. That is, to initialize a block, change a parameter, call an operation, and so on, a scenario calls an action on the event that contains the block instance.

Communication from the block to its host scenario is also accomplished by calling actions. In this case, the actions have been passed into the block by the scenario. For example, when a scenario initiates an operation the scenario passes in an action that the block must call to indicate that the operation has been completed.

Defining new blocks in Apama Studio

Apama Studio provides an integrated graphical environment for creating custom blocks that you can use to build scenarios in Event Modeler. The Apama Studio block editor contains two tabs, the Builder tab and the Source tab.

On the Builder tab, you add the metadata for the block and specify its interface. On the Source tab, you add the EPL code that implements the block's behavior. Apama Studio validates the EPL code you add to the block. When you save a block, Apama Studio saves it as a *block definition file* with a `.bdf`

extension. Block definition files are then used when you add the block to a scenario in the Event Modeler.

You can define a new block from scratch by using the block editor or you can base the new block on an existing event type definition.

See "File Definition Formats" on page 188 for detailed information on the internals of block definition files.

This topic is organized as follows:

Creating Blocks

Specifying the block metadata

Creating a block in Apama Studio consists of two main steps. In the first step you create the block metadata and specify its interface. In the second step you add the EPL code that implements the block's behavior.

When you create a new block, you should place it in the project's default blocks directory. This directory is found in the project's `catalogs` directory. The block directory has a name in the form *`project_name`*`\blocks`. So, for example, the default block directory of a project named `My_Project` will be `catalogs\My_Project blocks`. If you place the block in the default block directory, scenarios created in the project will automatically find them and make them available in Event Modeler when you are displaying the scenario.

You add a new block to a project by right-clicking the project and selecting New > Block from the pop-up context menu. Apama Studio displays the **New Block** wizard where you specify whether you want to create a block from scratch or base it on an existing event type. You also specify any other information that will make up the block's metadata.

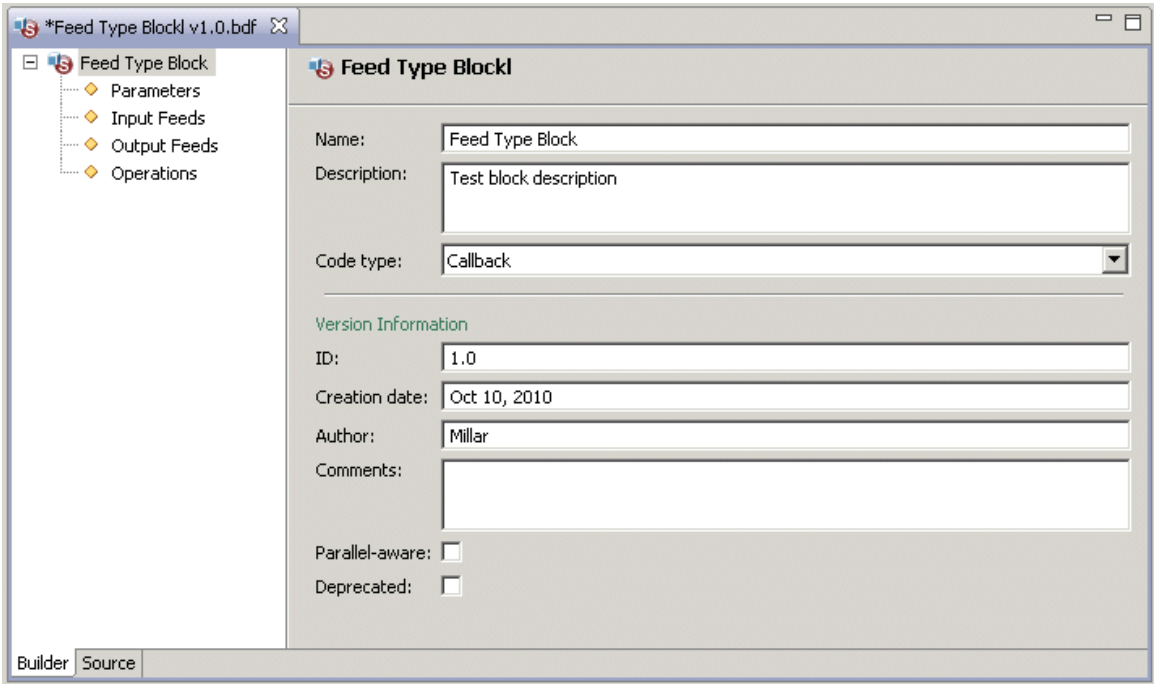
When you finish adding information in the **New Block** wizard, the block is added to the project and the block's metadata is displayed in the Builder tab of the block editor.

For specific steps on how to add a new block to an Apama project, see :

- "Creating a block with the block editor" on page 39
- "Creating a block from an EPL event definition" on page 40

Specifying the block interface

After you create a block, Apama Studio displays your new block in the Block Editor with the Builder tab selected:



Initially, the name of the block is selected and Apama Studio displays general information about the block. Most of the fields are self-explanatory and you can use them to help you maintain your blocks. Use the ID field to distinguish versions of your block. Select the Parallel-aware checkbox if you want to be able to use this block in a parallel scenario. See "Creating parallel-aware blocks" on page 85.

The Deprecated checkbox indicates whether this is an older version of the block. All Apama standard blocks that use the old-style block implementation (Apama releases prior to 4.2) are deprecated. They will not be supported in a future release.

If you have any custom blocks that use the old-style implementation, you should convert them to the new implementation and mark the old-style version as deprecated. To convert a block, open it in the Block Builder editor, select Callback or Callback (DEBUG) as the code type, and click the Source tab. See the Apama 5.0 migration guide for details about how you must manually edit the re-generated block file to correctly use the new implementation that Apama Studio generates for you. Apama Studio never automatically converts a block to use the new implementation.

Event Modeler uses the setting of the Deprecated checkbox to determine how to display the block in the Block Wiring panel. Deprecated blocks have an orange border while current blocks have a black border.

Also, suppose you write a custom block that uses the new-style implementation and you then revise that block. You can select the Deprecated checkbox for the older version to encourage use of the new version.

At this point, if you are creating a new block based on an existing event definition, the code for the block's input and output feeds, along with the fields associated with the feeds, and the block's operations has been generated.

If you are creating a new block from scratch, the block does not contain any of the parameters, input feeds, output feeds, and operations that provide the interface of the block. When you add these elements, Apama Studio generates the EPL code that defines the action that implements the element.

To add a parameter, input feed, output feed, or operation:

1. Right click the element you want to add and select Add Parameter, Add Input Feed, Add Output Feed, or Add Operation. The right side of the Builder tab displays the item's properties.
2. Fill in the values for the properties.
3. For input feeds and output feeds, right click the element and select Add Field.
4. In the Properties panel for the field you added in the previous step, fill in the values for the properties and field validation specifications.

When you save a block, Apama Studio generates the underlying code that defines the block's interface and saves it as a *block definition file* with a `.bdf` extension. To this file, you then add EPL code to implement the necessary behavior. To add code to the block, see "Adding EPL code to the block definition" on page 85.

Creating parallel-aware blocks

If you want a parallel scenario to use a block, you must mark that block as parallel-aware. You do this in the Builder tab of the block builder editor in Apama Studio. Select the block name. Then select the Parallel-aware checkbox near the bottom of the Builder tab fields.

The correlator runs each instance of a parallel scenario in a separate context. For information about contexts, see "Implementing parallel processing" in *Developing Apama Applications in EPL*.

When you mark a block you are creating as parallel-aware it means that you are taking responsibility for ensuring that the block functions correctly when run in multiple contexts. Blocks that do not listen for events are trivially parallel-aware since running in another context has no effect on that block. All of the block's interactions are mediated by the scenario.

Blocks that listen for events must ensure that the events they are listening for actually reach the context they are in. You can achieve this by storing a reference to the main context during the `instancePreSpawnInit()` action. Use this reference to inform services running in the main context where they should send events. Look at the Market Depth standard block for a good example of this.

Adding EPL code to the block definition

In Apama Studio, when you click the Source tab of the block builder editor, Apama Studio displays the block's definition file. Apama Studio generates and populates all XML elements including the `<code>` element. The `<code>` element contains the EPL code that specifies the block's behavior.

Apama Studio generates skeleton EPL with comments that indicate where to insert your code. The generated code defines the actions listed below. Each of these actions is a field in the event type that

defines the block. The block's scenario will call these actions to accomplish the work of the block. For each action that Apama Studio defines, you can add custom code that specifies the exact behavior you need.

- For each block parameter, there is an action that updates that parameter.
- For each block input feed, there is an action that takes as its arguments the fields of the feed.
- For each block operation, there is an action that performs the operation.
- For each block output feed, there is an action that takes as its arguments the fields of the feed.
- `setup` action
- `intancepreSpawnInit` action
- `instancePostSpawnInit` action
- `cleanup` action
- `start` action (for input blocks based on existing event definitions)
- `stop` action (for input blocks based on existing event definitions)
- `send` action (for output blocks based on existing event definitions)

In addition to defining these actions, Apama Studio generates sections for adding user-defined monitors, user-defined variables, and user-defined actions. Also, the generated EPL code defines a block-level variable named `blockInstanceId$`. This variable contains the integer that uniquely identifies the instance of the block among those owned by the containing scenario and all its instances.

To add EPL code to the block:

1. In Apama Studio, in the **Project Explorer** view, double-click the block's `.bdf` file.
2. In the block builder editor, click the Source tab.
3. On the Source tab, enter code as needed only where there is a white background.

   Code appears either with a gray background or a white background. Code with a gray background is maintained by Apama Studio and is not editable. The sections of code with a white background are the areas where you add your custom EPL code. Remember to remove the comment flags from lines on which you specify code.
4. Save the project.

As you add and edit code in your block, you have the full range of Apama Studio features as described in "Editing Apama files" on page 56. You also have the full range of navigating features as described in "Navigating in Apama files" on page 61.

Considerations for adding EPL code to the block definition

As you add custom code to your block, keep the following in mind:

- The `#` character denotes special names that will subsequently be assigned automatically by the code generator. Therefore, do not use the `#` character anywhere else in your EPL files, including within comments.

- You must not call `die()` anywhere in the block event type definition. Consequently you should not call `spawn()` in the block event type definition as you would have no way of terminating the new monitor instance.

In situations where you might want to spawn from within a block, use a utility monitor that is part of the block's definition instead. Insert the EPL code for a utility monitor in the `USER DEFINED MONITORS` section of your block definition file. For example, suppose your block subscribes to one or more market data feeds and you want to track data and status messages that result from each subscription. Write a utility monitor that listens for events related to the subscriptions and caches values that result from subscription operations. You can call `die()` in this monitor without affecting the block or the scenario.

- If the EPL code in your block causes a runtime error, for example you attempt a division by zero or you attempt to access an out of bounds index in a sequence or dictionary, the scenario monitor will be terminated by the correlator.

See also "Timeliness of acknowledgements" on page 93.

Details about EPL code that you can add

The following sections describe what Apama Studio generates for you and where to add EPL code:

Actions that update parameters

Apama Studio generates skeleton code for an action for each parameter you specify for the block. Each action updates the value of the parameter. These actions are named `update$`*`parameter_name`*, where *`parameter_name`* is the metadata name you specified for the parameter. Each action takes the parameter's specified name as an argument.

Each time the value of a block parameter changes in the scenario, the scenario calls the corresponding update action on the block. It is up to you to define appropriate EPL code in the body of this action to handle the block parameter update. It is bad practice to send updates to output feeds during a parameter update action because it can cause unexpected results in the running scenario.

If a parameter should not be editable, leave the body of its update action empty.

For example, if a block specifies a `string` parameter called `New Parameter 1` Apama Studio generates the following skeleton code:

```
    action update$new_parameter_1(string new_parameter_1) {
// BLOCKBUILDER - USER DEFINED ACTION
//
// -- insert handler for modifications to new parameter 1 --
//
// BLOCKBUILDER - END OF USER DEFINED ACTION
    }
```

If a block is based on an existing event and is an input block, the skeleton code contains additional information about the parameter. In the following example a parameter based on the event field `customerName` has been specified:

```
    action update$customerName(string customerName) {
// BLOCKBUILDER - USER DEFINED ACTION
      parameter_customerName := customerName ;
      isSet_parameter_customerName := true;
      setupNewListener();
// BLOCKBUILDER - END OF USER DEFINED ACTION
```

Actions that update input feeds

Apama Studio generates skeleton code for an action for each input feed you specify for the block. Each action updates the values of the corresponding input feed's fields. These actions are named `input$`*`input_feed_name`*, where *`input_feed_name`* is the metadata name you specified for the input feed. Each action takes an argument for each field in the corresponding input feed.

It is up to you to define appropriate EPL code in the body of this action to handle the update to the input feed. For example, if a block specifies an input feed named `Input Feed 1`, Apama Studio generates the following skeleton code:

```
   action input$new_input_feed_1(#string string_field, float float_field)
{
// BLOCKBUILDER - USER DEFINED ACTION
//
// -- insert handler for new input events on stream new input feed 1 --
//
// BLOCKBUILDER - END OF USER DEFINED ACTION
   }
```

Actions that perform operations

Apama Studio generates skeleton code for an action for each operation that you specify in the block. Each action performs the operation. These actions are named `operation$`*operation_name* where *operation_name* is the metadata name you specified for the operation. Each action takes only an `acknowledge()` action variable argument.

It is up to you to define appropriate EPL code in the body of this action to handle the operation's invocation. You must call the `acknowledge()` action when the operation is complete. There are constraints on how long you can hold up a call to `acknowledge()`. Often, an operation updates output feeds before calling `acknowledge()`. See "Timeliness of acknowledgements" on page 93.

For example, if a block specifies an operation called `New Operation 1`, Apama Studio generates the following skeleton code:

```
    action operation$new_operation_1(action<> acknowledge) {
// BLOCKBUILDER - USER DEFINED ACTION
//
// -- insert handler for invocations of operation new operation 1 --
//
// BLOCKBUILDER - END OF USER DEFINED ACTION
    }
```

Actions that update output feeds

Apama Studio generates a `sendOutput` action for each output feed that you specified for the block. Each action updates the values of the corresponding output feed's fields. These actions are named `sendOutput$`*output_feed_name* where *output_feed_name* is the metadata name of the output feed. Each action takes an argument for each field in the corresponding output feed.

You do not need to add code for these actions. To output results from your block you should call one of these output feed actions. If your block uses an output feed `new_output_feed_1` with a boolean field new_field_1 and a string field new_field_2, Apama Studio generates the following code:

```
action<boolean,string> sendOutput$new_output_feed_1;
```

setup action

Apama Studio generates skeleton code for the `setup()` action. The scenario calls the `setup()` action once on each block instance in a scenario definition. The scenario makes this call when you inject the scenario into the correlator. Use the `setup()` action to specify any initialization that is not specific to a scenario instance. Apama Studio generates the following skeleton code:.

```
    action setup {
// BLOCKBUILDER - USER DEFINED ACTION
//
// -- insert setup code --
//
// BLOCKBUILDER - END OF USER DEFINED ACTION
    }
```

instancePreSpawnInit action

Apama Studio generates skeleton code for the `instancePreSpawnInit()` action. The scenario calls the `instancePreSpawnInit()` action on each scenario instance. The scenario makes this call just before it spawns the scenario instance. The scenario passes the following values into the `instancePreSpawnInit()` action:

- Scenario ID
- Dictionary of extra data
- Target context the scenario instance will run in. For a scenario that is not parallel (that is, it is a serial scenario), the target context is always the main context.

Use the `instancePreSpawnInit()` action to perform initialization in the main context. For example, the main context might need information about which context the scenario instance, and therefore the block instance(s) will run in. A block cannot generate output feed values inside the `instancePreSpawnInit()` action, but it can generate output feed values inside the `instancePostSpawnInit()` action.

When the scenario calls the `instancePreSpawnInit()` action, it passes an `acknowledgment()` action. You are responsible for ensuring that the `instancePreSpawnInit()` action calls this `acknowledgment()`

action when it has completed this phase of initialization. To help you do this, Apama Studio generates a call to `acknowledge()` when it generates the skeleton code for the block. See "Timeliness of acknowledgements" on page 93.

Apama Studio generates the following skeleton code:

```
       action instancePreSpawnInit (
           integer blockInstanceId$,
           string scenarioId$,
           dictionary<string, string> userData$,
           context target,
           action<> acknowledge) {
       self.blockInstanceId$ := blockInstanceId$;
// BLOCKBUILDER - USER DEFINED ACTION
//
// -- insert pre-spawn initialisation code --
//
       acknowledge();
// BLOCKBUILDER - END OF USER DEFINED ACTION
    }
```

If a block is based on an existing event, the skeleton code contains additional code to specify the context.

```
   action instancePreSpawnInit(
        integer blockInstanceId$,
        string scenarioId$,
        dictionary<string, string> userData$,
        context target,
        action<> acknowledge) {
    self.blockInstanceId$ := blockInstanceId$;
// BLOCKBUILDER - USER DEFINED ACTION
//
// -- insert pre-spawn initialisation code --
//
preSpawnContext := context.current();
      acknowledge();
// BLOCKBUILDER - END OF USER DEFINED ACTION
    }
```

instancePostSpawnInit action

Apama Studio generates skeleton code for the `instancePostSpawnInit()` action. The scenario calls the `instancePostSpawnInit()` action on each newly spawned scenario instance. The scenario makes this call right after it spawns the scenario instance. Part of this action is to pass the following to the scenario instance:

- Scenario ID
- Dictionary of extra values
- Initial values of the block's parameters
- Additional data for use by the automatically generated code.

When the scenario calls the `instancePostSpawnInit()` action, it passes an `acknowledgment()` action. You are responsible for ensuring that the `instancePostSpawnInit()` action calls this `acknowledgment()` action when it has completed this phase of initialization. To help you do this, Apama Studio generates a call to `acknowledge()` when it generates the skeleton code for the block. See "Timeliness of acknowledgements" on page 93.

Apama Studio generates the following skeleton code:

```
       action instancePostSpawnInit (
           integer blockInstanceId$,
           string ownerId$,
```

```
           string scenarioId$,
           dictionary<string, string> userData$,
           action<> acknowledge)
           param_type param1 //one for each parameter
           action<output_field_types> sendOutput$outfeed   {
                // one action like the above for each output feed
                // one line like the following for each output feed
       self.sendOutput$outfeed1 := sendOutput$outfeed1;
// BLOCKBUILDER - USER DEFINED ACTION
//
// -- insert post-spawn initialisation code --
//
       acknowledge();
// BLOCKBUILDER - END OF USER DEFINED ACTION
    }
```

If the block is an input block based on an existing event, the generated code looks like this:

```
   action instancePostSpawnInit(
     integer blockInstanceId$,
     string ownerId$,
     string scenarioId$,
     dictionary<string, string> userData$,
     action<> acknowledge,
     string name,
     action<string,float> sendOutput$TestEvent) {
         self.sendOutput$TestEvent := sendOutput$TestEvent;
// BLOCKBUILDER - USER DEFINED ACTION
//
// -- insert post-spawn initialisation code --
//
          enqueue TestEventForwardRequest (context.current()) to preSpawnContext;
          // Store the initial values
          parameter_name := name;
          acknowledge();
// BLOCKBUILDER - END OF USER DEFINED ACTION
   }
```

If the block is an output block based on an existing event, the generated code looks like this:

```
      action instancePostSpawnInit(
        integer blockInstanceId$,
        string ownerId$,
        string scenarioId$,
        dictionary<string, string> userData$,
        action<> acknowledge,
        string name) {
// BLOCKBUILDER - USER DEFINED ACTION
//
// -- insert post-spawn initialisation code --
//
        if (preSpawnContext.getId() = context.current().getId()) then {
           serialExecution := true;
        }
        else {
           serialExecution := false;
        }
        acknowledge();
// BLOCKBUILDER - END OF USER DEFINED ACTION
     }
```

The scenario is not in a fully created state until all blocks have acknowledged their `instancePostSpawnInit()` call. Also, updating of output feeds is not supported at any stage before `instancePostSpawnInit()` is called. If initial values for output feeds need to be generated, do this in the `instancePostSpawnInit()` action.

cleanup action

Apama Studio generates skeleton code for the `cleanup()` action. When a block's scenario enters its end state, is deleted, or dies for some other reason, the scenario calls the block's `cleanup()` action. Even if there is a runtime error, the scenario calls the `cleanup()` action.

After the scenario calls the `cleanup()` action, the block should no longer try to update its output feeds. The block should act in every possible way as if it was dead. However, if there is any finalization work that you want to accomplish, you can add it to the body of the `cleanup()` action. Apama Studio generates the following skeleton code:

```
    action cleanup {
// BLOCKBUILDER - USER DEFINED ACTION
//
// -- insert finalization code --
//
// BLOCKBUILDER - END OF USER DEFINED ACTION
    }
```

start action

For blocks that are based on existing event definitions and are specified as input blocks, Apama Studio generates code for a `start` action. Once the `start` operation is invoked the block calls a `setupNewListener` action, which creates the listener code for the event on which the block is based. If any event fields have been specified when defining the block, they are used as parameters to create filters in the listener. Apama Studio generates the following code:

```
  action operation$start(action<> acknowledge) {
// BLOCKBUILDER - USER DEFINED ACTION
    isStarted := true;
    setupNewListener();
    acknowledge();
// BLOCKBUILDER - END OF USER DEFINED ACTION
  }
```

stop action

For blocks that are based on existing event definitions and are specified as input blocks, Apama Studio generates code for a `stop` action. When the `stop` operation is invoked all active listeners are terminated. Apama Studio generates the following code (where "`l`" is a `listener` defined elsewhere):

```
  action operation$stop(action<> acknowledge) {
// BLOCKBUILDER - USER DEFINED ACTION
    isStarted := false;
    l.quit();
    acknowledge();
// BLOCKBUILDER - END OF USER DEFINED ACTION
  }
```

send action

For blocks that are based on existing event definitions and are specified as output blocks, Apama Studio generates code for a `send$`*`event`* action. When the `send` action is called, it sends the specified event on which the block is based. For example, with a specified event, `testEvent` (containing two fields, `name` and `IDnum`), Apama Studio generates the following code:

```
    action operation$send$_testEvent(action<> acknowledge) {
// BLOCKBUILDER - USER DEFINED ACTION
//
// -- insert handler for invocations of operation send_testEvent --
//
      if (serialExecution) then{
         route  testEvent(parameter_name,parameter_IDnum);
```

```
      }
      else {
          enqueue testEvent(parameter_name,parameter_IDnum) to preSpawnContext;
      }
 acknowledge();
// BLOCKBUILDER - END OF USER DEFINED ACTION
     }
```

User-defined monitors or event types

If you need to add monitors or event types to a block, define them in the specified section of the block's generated EPL code:

```
// BLOCKBUILDER - USER DEFINED MONITORS
//
// -- insert any additional monitors you require --
//
// BLOCKBUILDER - END OF USER DEFINED MONITORS
```

For more information on designing monitors see "Defining Monitors" in *Developing Apama Applications in EPL*.

User-defined variables

If you need to add variables to a block, define them in the specified section of the block's generated EPL code, which is the first section in the block's `#block#` event code:

```
event #block# {
// BLOCKBUILDER - USER DEFINED VARIABLES
//
// -- insert any additional variables you require --
//
// BLOCKBUILDER - END OF USER DEFINED VARIABLES
```

User-defined actions

In addition to the actions described above you can add any other actions that you require to implement the unique functionality of your block. Add additional actions at the end of the block definition file in the specified section:

```
// BLOCKBUILDER - USER DEFINED ACTIONS
//
// -- insert any additional actions required --
//
// BLOCKBUILDER - END OF USER DEFINED ACTIONS
```

Timeliness of acknowledgements

When a scenario calls an action that takes an `acknowledgement()` action parameter the scenario expects to receive a timely acknowledgement.

This means that the acknowledgement must be made within the chain of routed events that are currently being processed, starting with the event that is the immediate cause of the operation being performed. This constraint exists because the scenario is in a state of limbo while it is waiting for an acknowledgement. If another event comes into the scenario, either a control event or one that comes into one of its blocks, while the scenario is waiting for an acknowledgement then the scenario can get into an inconsistent state. For example, during a block operation, the scenario expects updates only from the block that the operation is called on.

This constraint is usually easily met. If an operation routes a request event that it expects a routed response to then the block can simply wait for that response before returning the acknowledgement to the scenario. Alternatively, the block can set up a `completed` listener for the request event. If the block does not expect a response with interesting data that it wants to reflect to output feeds then the block can immediately return the acknowledgement even if there are still routed events to be processed. It is especially important to ensure that all operations are acknowledged for all paths through the code because unacknowledged operations will cause the scenario to hang.

An example block

As an example, consider the Correlation Calculator Block, which is one of the standard blocks provided with Apama.

The Correlation Calculator Block calculates the correlation coefficient between two streams of data. The calculation can be performed over an unlimited set of data from each stream, or a set limited by number of samples or age of samples. The calculator generates output only if there is at least one suitable sample from each stream.

A correlation coefficient approaching +1.0 shows a strong correlation between the streams, a coefficient close to 0.0 shows little or no correlation between the streams and a coefficient approaching –1.0 shows an inverse correlation between the streams; for example, if one is increasing, the other is decreasing.

The following topics describe the Correlation Calculator block:

- "Description of the Correlation Calculator block interface" on page 94
- "Description of the Correlation Calculator block EPL" on page 96

Creating Blocks

Description of the Correlation Calculator block interface

The Correlation Calculator block has the following parameters:

Parameter	Description
`period`	The maximum age of any sample that is used in the calculations, in seconds. Any samples older than this will be discarded before performing the calculation.
`size`	The maximum number of samples per stream that are used in the calculation.

One or both of the above parameters must be `0`, in which case that limit is not imposed. It is not possible to restrict the number of samples by both age and number of samples, but it is possible to remove the limit on the number of samples (thus an unbounded set of samples is kept). Note that imposing a limit after input events have been received will clear all existing samples.

The Correlation Calculator block has the following operations:

Operation	Description
`start`	Starts the calculation of coefficients. Must be called before the calculator will generate any statistics (output feed).
`stop`	Stops the calculation of further coefficients. Any subsequent events on the input feeds are ignored.
`clear`	Discards all current data.

The Correlation Calculator block defines the following input feeds, each with one field:

Input feed	Fields	Description
`data1`	`value`	The first input set.
`data2`	`value`	The second input set.

Note that at least one value from each feed must have been received (and if set, within `period` seconds) before an output will be generated.

The Correlation Calculator block has the following output feed:

Output feed	Fields	Description
`statistics`	`correlation`	The correlation coefficient (between `-1.0` and `+1.0`).
	`samples`	The number of sample pairs used for this calculation.

The XML elements at the beginning of the Correlation Calculator's block definition file describe this interface. When you create your own block, Apama Studio generates and populates these XML elements for you.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE block SYSTEM "http://www.apama.com/dtd/bdf.dtd">
<!--Apama Block Definition File-->
<block name="Correlation Calculator">
  <version>
    <id>2.0</id>
    <date>7 May 2009</date>
    <author>Rune Madsen</author>
    <comments>Copyright(c) 2013 Software AG, Darmstadt, Germany and/or its licensors</comments>
  </version>
  <description>Calculates the correlation of two input data streams over a configurable
    time window and sample set size.</description>
  <properties parallel-aware="true" deprecated="false">
    <input-feeds>
      <feed name="data1" id="9578163894100102">
        <description>The first stream of numeric data to use in the correlation
          calculations</description>
        <field name="value" id="9578163894100103">
          <description>The numeric data value</description>
          <validation type="float" stringcase="mixed" trim="true" unique="false"
            mutability="mutable" />
        </field>
      </feed>
      <feed name="data2" id="9578163894100104">
        <description>The second stream of numeric data to use in the correlation
          calculations</description>
        <field name="value" id="9578163894100105">
```

```
        <description>The numeric data value</description>
        <validation type="float" stringcase="mixed" trim="true" unique="false"
          mutability="mutable" />
      </field>
    </feed>
  </input-feeds>
  <output-feeds>
    <feed name="statistics" id="9578163894100106">
      <description>Stream of correlation values generated every time a new data item
        arrives</description>
      <field name="correlation" id="9578163894100107">
        <description>The correlation of the samples in the data sets. Between -1 and
          +1.</description>
        <validation type="float" stringcase="mixed" trim="true" unique="false"
          mutability="mutable" />
      </field>
      <field name="samples" id="9578163894100108">
        <description>The number of sample pairs used in the correlation
          calculation</description>
        <validation type="integer" stringcase="mixed" trim="true" unique="false"
          mutability="mutable" />
      </field>
    </feed>
  </output-feeds>
  <parameters>
    <field name="period" id="9578163894100109">
      <description>The duration of the configurable time window given in seconds.
        Samples older than the period will be discarded from the data set. Set to zero
        to keep samples indefinitely, up to the maximum number of samples specified
        with the size parameter.</description>
      <validation type="float" stringcase="mixed" trim="true" unique="false"
        mutability="mutable" />
    </field>
    <field name="size" id="9578163894100110">
      <description>The maximum size of the sample set. The oldest sample will be
        replaced by the new sample when the total number of samples has reached this
        limit. Set to zero to keep all samples, unless period is set.</description>
      <validation type="integer" stringcase="mixed" trim="true" unique="false"
        mutability="mutable" />
    </field>
  </parameters>
  <operations>
    <operation name="start" id="9578163894100111">
      <description>Activate the correlation calculations</description>
    </operation>
    <operation name="stop" id="9578163894100112">
      <description>Pause the correlation calculations</description>
    </operation>
    <operation name="clear" id="9578163894100113">
      <description>Clear the existing sample data</description>
    </operation>
  </operations>
</properties>
```

Description of the Correlation Calculator block EPL

After the XML elements that describe the block interface, there is a `<code>` element. The `<code>` element contains the EPL. The first section in which you can add custom EPL code is the user-defined monitors section. The Correlation Calculator block defines a few events here.

User-defined monitors and/or events

```
<code><![CDATA[// Apama generated code - ONLY EDIT INDICATED SECTIONS
// Generated code type: CALLBACK
// Generated code version: 1
```

```
// BLOCKBUILDER - USER DEFINED MONITORS

event CorrelationCalculator_DataPoint {
  float value1;
  float value2;
  float time;
}

event CorrelationCalculator_Incr {
  float x1;
  float y1;
  float x2;
  float y2;
  float xy;
  float N;
}

event CorrelationCalculator_InputData {
  float value;
  float time;
}

// BLOCKBUILDER - END OF USER DEFINED MONITORS
```

User-defined variables

After the section for user-defined monitors or events, Apama Studio begins the event type definition that implements the block. The placeholder name of the event type is always `#block#`. When you inject a scenario that uses a block, the correlator replaces `#block#` with the actual name of the block plus a unique number that distinguishes the instance of the block from other instances.

The first section after the event declaration is for user-defined variables. Each variable is a field in the event type. The Correlation Calculator block defines a number of variables.

```
event #block# {
// BLOCKBUILDER - USER DEFINED VARIABLES
  sequence<CorrelationCalculator_DataPoint> dataset;
  boolean running;
  boolean infinite;
  CorrelationCalculator_Incr incr;

  integer MAX_INT;
  float MAX_FLOAT;
  float NO_CORRELATION;

  CorrelationCalculator_InputData inputdata1;
  CorrelationCalculator_InputData inputdata2;

  float period;
  integer size;

// BLOCKBUILDER - END OF USER DEFINED VARIABLES
```

Actions for updating output feeds

Following the user-defined variables are the variables that Apama Studio automatically generates for every block. This includes an `integer` variable to contain the block instance ID and an `action` variable for each output feed in the block. For the Correlation Calculator block, these variables are defined as follows:

```
  integer blockInstanceId$;
  action<float,integer> sendOutput$statistics;
```

Actions for updating parameters

Next come the actions that update parameters. Apama Studio defines the action and the block writer fills in the code that actually updates the parameter. For the Correlation Calculator block, the following actions update the `period` and `size` parameters:

```
  action update$period(float period) {
// BLOCKBUILDER - USER DEFINED ACTION
    self.period := period;
    updateInfinite();
// BLOCKBUILDER - END OF USER DEFINED ACTION
  }

  action update$size(integer size) {
// BLOCKBUILDER - USER DEFINED ACTION
    self.size := size;
    updateInfinite();
// BLOCKBUILDER - END OF USER DEFINED ACTION
  }
```

Actions for updating input feeds

Next come the actions that update input feeds. Again, Apama Studio defines the action and the block writer fills in the code that actually does the update. For the Correlation Calculator block, the following actions update the `data1` and `data2` input feeds:

```
  action input$data1(float value) {
// BLOCKBUILDER - USER DEFINED ACTION
     if not running then {
       return;
     }
    self.inputdata1.value := value;
    self.inputdata1.time := currentTime;
    doStats1();
// BLOCKBUILDER - END OF USER DEFINED ACTION
  }

  action input$data2(float value) {
// BLOCKBUILDER - USER DEFINED ACTION
    if not running then {
      return;
    }
    self.inputdata2.value := value;
    self.inputdata2.time := currentTime;
    doStats2();
// BLOCKBUILDER - END OF USER DEFINED ACTION
  }
```

Actions for performing operations

The actions that perform operations come next. For the Correlation Calculator block, these actions are defined as follows:

```
  action operation$start(action<> acknowledge) {
// BLOCKBUILDER - USER DEFINED ACTION
    running := true;
    acknowledge();
// BLOCKBUILDER - END OF USER DEFINED ACTION
  }

  action operation$stop(action<> acknowledge) {
// BLOCKBUILDER - USER DEFINED ACTION
    running := false;
    acknowledge();
// BLOCKBUILDER - END OF USER DEFINED ACTION
  }
```

```
  action operation$clear(action<> acknowledge) {
// BLOCKBUILDER - USER DEFINED ACTION
    inputdata1.value := MAX_FLOAT;
    inputdata2.value := MAX_FLOAT;
    dataset.setSize(0);
    incr := new CorrelationCalculator_Incr;
    acknowledge();
// BLOCKBUILDER - END OF USER DEFINED ACTION
  }
```

Standard setup and cleanup actions

After defining the actions that implement the interface to the block, Apama Studio defines the standard setup and cleanup actions that it defines in every block. These look like the following for the Correlation Calculator block. Notice that the `instancePreSpawnInit()` action has no user-defined code. The scenario calls this action on each new scenario instance. Since nothing other than what Apama Studio automatically fills in is necessary, the user-defined section for the `instancePreSpawnInit()` action is empty.

```
  action setup {
// BLOCKBUILDER - USER DEFINED ACTION
    MAX_INT := 0x7fffffffffffff;
    MAX_FLOAT := 1.0e300;
    NO_CORRELATION := -2.0;
// BLOCKBUILDER - END OF USER DEFINED ACTION
  }

  action instancePreSpawnInit(integer blockInstanceId$,
      string scenarioId$,
      dictionary<string, string> userData$,
      context target,
      action<> acknowledge) {
    self.blockInstanceId$ := blockInstanceId$;
// BLOCKBUILDER - USER DEFINED ACTION
//
// -- insert pre-spawn initialisation code --
//
    acknowledge();
// BLOCKBUILDER - END OF USER DEFINED ACTION
  }

  action instancePostSpawnInit(integer blockInstanceId$,
      string ownerId$,
      string scenarioId$,
      dictionary<string, string> userData$,
      action<> acknowledge,
      float period,
      integer size,
      action<float,integer> $$sendOutput$statistics) {
    self.$$sendOutput$statistics := $$sendOutput$statistics;
// BLOCKBUILDER - USER DEFINED ACTION
    self.period := period;
    self.size := size;
    inputdata1.value := MAX_FLOAT;
    inputdata2.value := MAX_FLOAT;
    updateInfinite();
    acknowledge();
// BLOCKBUILDER - END OF USER DEFINED ACTION
  }

  action cleanup {
// BLOCKBUILDER - USER DEFINED ACTION
// BLOCKBUILDER - END OF USER DEFINED ACTION
  }
```

User-defined actions

Finally, any additional user-defined actions come at the end of the block definition file. For the Correlation Calculator block, these actions contain the unique functional content of this block.

```
// BLOCKBUILDER - USER DEFINED ACTIONS
  action doStats1 {
    if inputdata2.value != MAX_FLOAT then {
      doStatsCommon(inputdata2.time);
    }
  }

  action doStats2 {
    if inputdata1.value != MAX_FLOAT then {
      doStatsCommon(inputdata1.time);
    }
  }

  action doStatsCommon(float timestamp) {
    float N;
    float Mx;
    float sum, div;
    float correlation;

    if not infinite then {
      // Remove expired samples
      removeExpiredSamples();

      // Add new pair to dataset
      dataset.append(
        CorrelationCalculator_DataPoint(inputdata1.value,
            inputdata2.value, timestamp));
    }
    incrAdd(inputdata1.value, inputdata2.value);

    // Calculate correlation
    N := incr.N;
    Mx := incr.x1 / N;
    sum := incr.xy - Mx*incr.y1;
    div := (incr.x2 - Mx*incr.x1) * (incr.y2 - incr.y1*incr.y1/N);
    if sum = 0.0 then {
      correlation := 0.0;
    } else
    if div != 0.0 then {
      correlation := sum / div.sqrt();
    } else {
      correlation := NO_CORRELATION;
    }
    sendOutput$statistics(correlation, N.floor());
  }

  action removeExpiredSamples {
    float timeLimit := -MAX_FLOAT;
    integer sizeLimit := MAX_INT;
    if self.period > 0.0 then {
      timeLimit := currentTime - self.period;
    } else
    if self.size > 0 then {
      sizeLimit := self.size;
    }
    while (dataset.size() > 0 and dataset[0].time <= timeLimit)
      or dataset.size() >= sizeLimit {
      incrRemove(dataset[0].value1, dataset[0].value2);
      dataset.remove(0);
    }
  }

  action updateInfinite {
    boolean wasInfinite := infinite;
```

```
    // Set infinite to true if period/size is infinite
    infinite := self.period <= 0.0 and self.size <= 0;
    if infinite then {
      dataset.setSize(0);
    } else
    if wasInfinite then {
      // Infinite has gone from true to false,
      // must reset incremental data
      incr := new CorrelationCalculator_Incr;
    }
  }

  action incrAdd(float x, float y) {
    incr.x1 := incr.x1 + x;
    incr.y1 := incr.y1 + y;
    incr.x2 := incr.x2 + x*x;
    incr.y2 := incr.y2 + y*y;
    incr.xy := incr.xy + x*y;
    incr.N  := incr.N + 1.0;
  }

  action incrRemove(float x, float y) {
    incr.x1 := incr.x1 - x;
    incr.y1 := incr.y1 - y;
    incr.x2 := incr.x2 - x*x;
    incr.y2 := incr.y2 - y*y;
    incr.xy := incr.xy - x*y;
    incr.N  := incr.N - 1.0;
  }
// BLOCKBUILDER - END OF USER DEFINED ACTIONS
}]]></code>
</block>
```

Chapter 4: Launching Projects

As you develop your applications, you can test them by running them from Apama Studio. Projects contain *launch configurations* that specify which resources in the project need to be started and any initialization information they need. By default, a single launch configuration is created for each project the first time it is launched, but you can create others to test an application with a different set of monitors or a different version of a block.

This section contains the following topics:

- "Running Apama projects" on page 102
- "Monitoring apama applications" on page 115

For information about creating launch configurations for debugging and profiling purposes, see the following:

- "Debugging EPL Applications" on page 125
- "Debugging JMon Applications" on page 134
- "Profiling EPL Applications" on page 146

Running Apama projects

Apama Studio uses a project's launch configuration to start the application. You can use the project's default launch configuration or you can define one or more configurations for launching your Apama project in different ways.

Default launch configuration

An application's default launch configuration starts the default correlator and uses all the monitors and events defined in the project. A project's default launch configuration is available in both the **Workbench** and **Developer** perspectives.

Workbench perspective

To start the default launch configuration for the current project from the **Workbench** perspective:

1. In the **Workbench Project** view, click the Start button  . This starts the default correlator with all the project's monitors, events and scenarios. If the project includes a default dashboard, Apama Studio launches it in the Dashboard Viewer. Information about the running application is shown in the **Console** view and if the project has a scenario, the **Scenario Browser** view is displayed.

2. To restart the application, click the Restart button  in the**Workbench Project** view.
3. To stop the click the Stop button  in the**Workbench Project** view. In addition to halting the application, this closes the Dashboard Viewer

Developer perspective

To start the default launch configuration for a project from the **Developer** perspective:

1. In the **Project Explorer** view, select the project you want to run.
2. Select Run > Run from the Apama Studio menu. This starts the default correlator with all the project's monitors, events and scenarios. If the project includes a default dashboard, Apama Studio launches it in Dashboard Viewer. Information about the running application is shown in Apama Studio's **Console** view.
3. If you want to inspect the behavior of the project's scenarios in the Scenario Browser, select Window > Show View > Scenario Browser.
4. To stop a running application click the Terminate button  in the **Console** view. If the launch configuration started Dashboard Viewer, you may want to stop Dashboard Viewer before stopping the application otherwise when you stop the application Dashboard Viewer will display a message its connection to the correlator was lost.

Defining custom launch configurations

In many cases you may want to create custom launch configurations, for example to run your applications with a subset of the monitors in your project or if your application relies on an IAF adapter. Note that the **Apama Workbench** perspective is targeted at creation of simple applications and so does not provide much support for projects with multiple launch configurations. If you need the power of multiple launch configurations use the **Apama Developer** perspective.

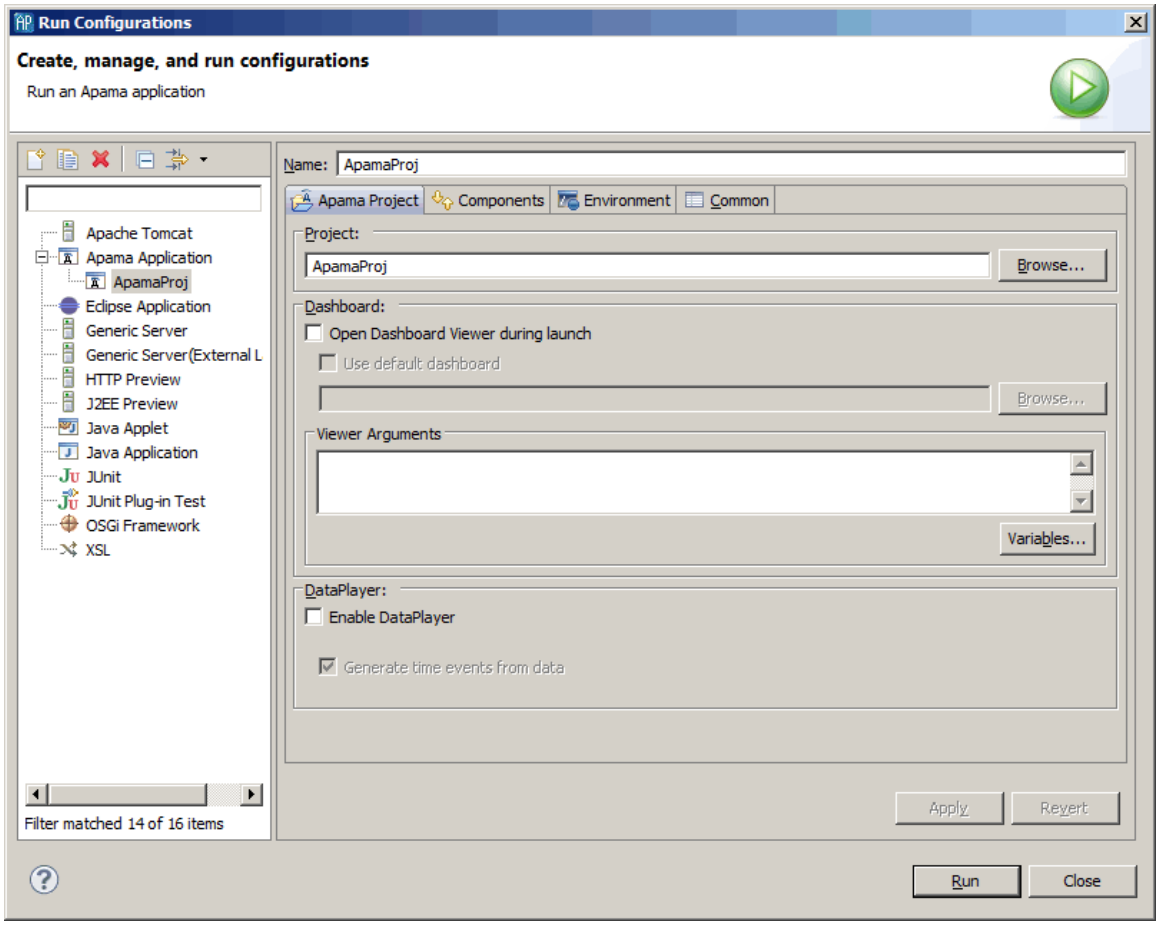
When you create a new shared custom launch configuration Apama Studio creates two files and places them in the directory specified by Shared file. The names of these files have the following format and the launch configuration information is split between them:

- *launch_config_name*.deploy
- *launch_config_name*.launch

Any changes you make to the launch configuration will be reflected in the `.deploy` file. However, if you export a launch configuration to an Ant deployment script and then the shared location changes then you must re-export the launch configuration to a new Ant deployment script. If you do not, the old Ant deployment script fails because it cannot find the `.deploy` file. See

To create a launch configuration:

1. In the **Project Explorer** view, select the project you want.
2. If you are using the **Apama Developer** perspective, select Run > Run Configurations from the Apama Studio menu. The **Create, manage, and run configurations** wizard starts. The wizard has four tabs, the Apama Project tab, the Components tab, the Environment tab, and the Common tab.



If you are using the **Apama Workbench** perspective, select the project of interest in the **Workbench Project** view and then click the Edit button just above the Start, Stop, and Restart buttons in the launch control panel. This allows the default launch configuration for the project to be edited. Creating multiple launch configurations is not recommended for **Workbench** perspective users

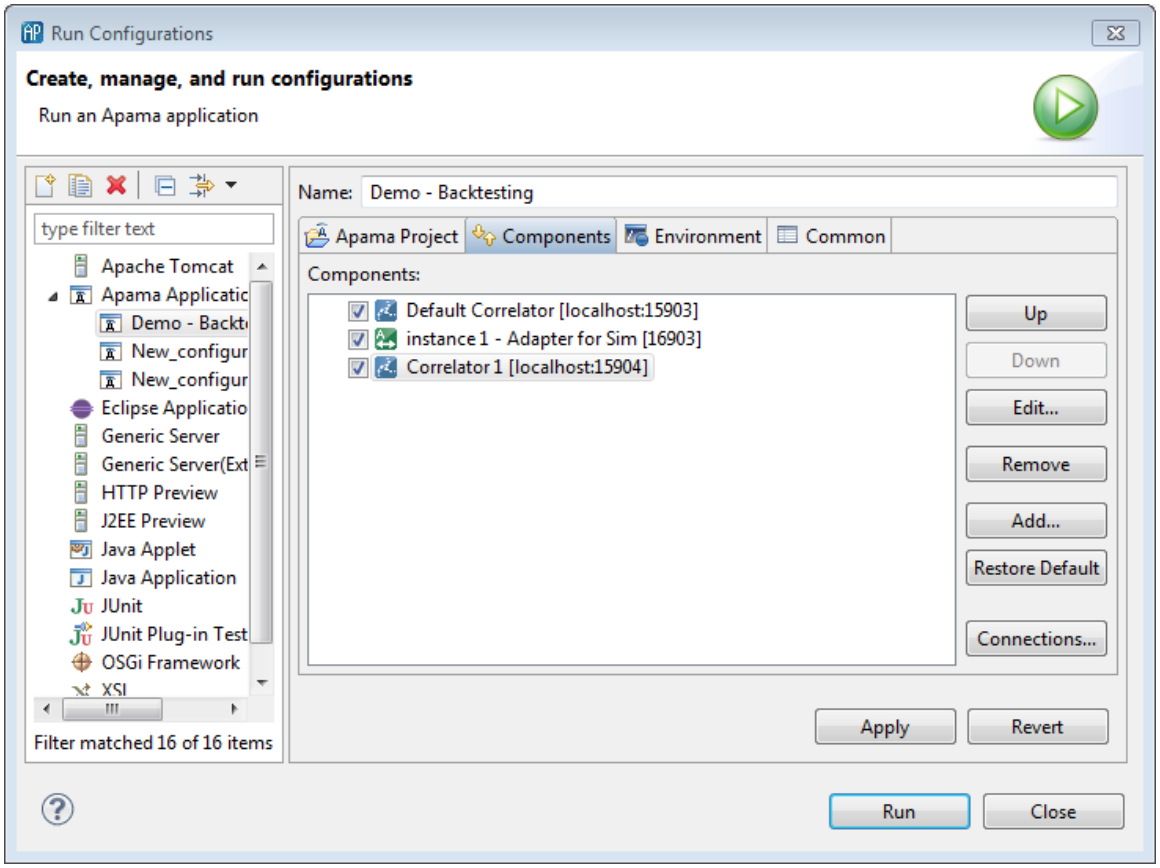
3. In the wizard's Name field assign a name to the launch configuration
4. On the Apama Project tab fill in the following information:
   - Project — The project field specifies the project to launch.
   - Dashboards — This contains information about launching a dashboard when the project runs. It contains the following fields:

   Open Dashboard Viewer during launch — Specifies whether or not the Dashboard Viewer should be launched when running the project. The default is to run the Dashboard Viewer.

   Use default dashboard — The default dashboard is the dashboard project in the `dashboards` directory of the project. You can launch another dashboard by disabling the default dashboard and specifying the project relative path of the dashboard to launch.

   viewer Arguments — Specify any arguments to be added to the end of the command line for the dashboard viewer process.
5. The Components tab lists the components that are needed by the project such as additional correlators or external processes. You can add and remove components, edit their specifications, change the order in which they are started, or specify connections between correlators. Components are started in the order

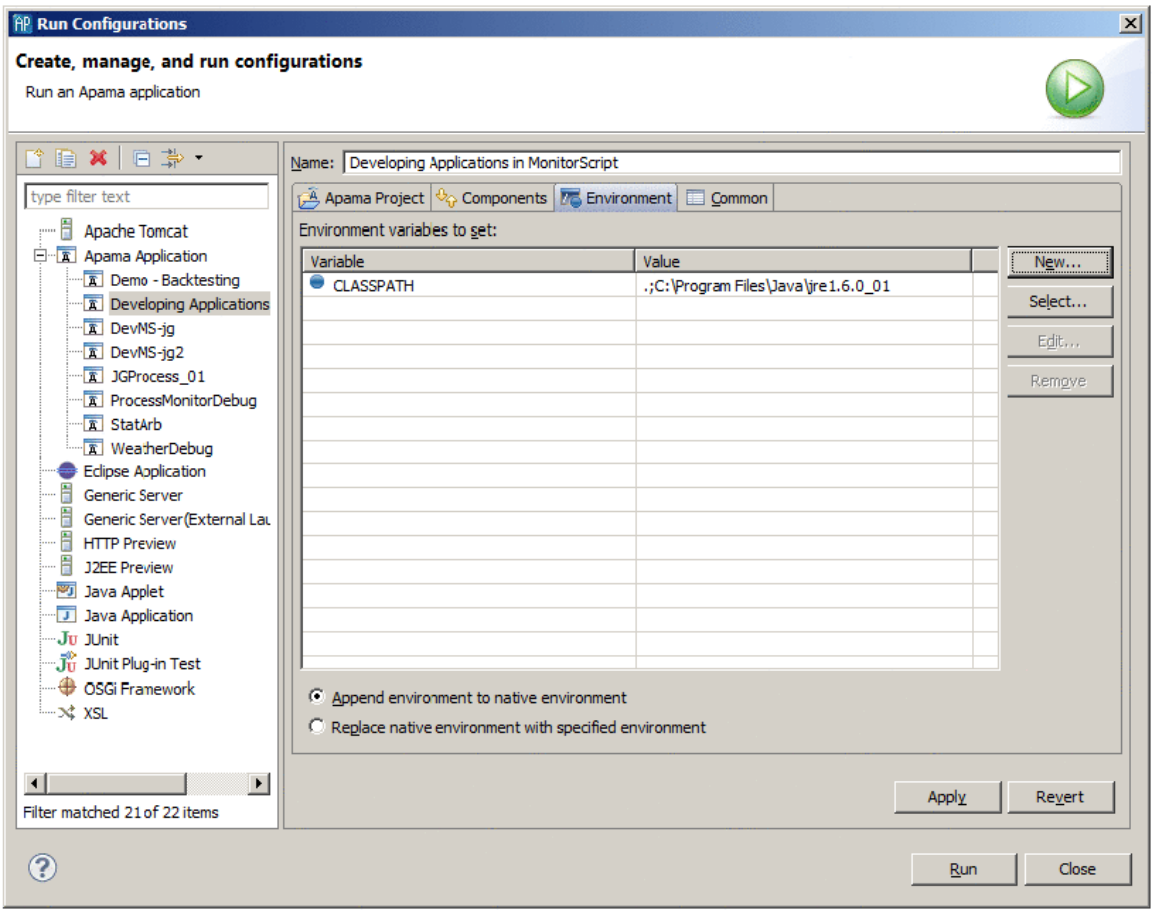
in which they appear in the Components tab from top to bottom. However, there is no waiting for one component to finish its startup before the next component is started. In other words, you cannot depend on startup for one component to already be complete when a subsequent component is started.



You specify information on this tab as follows:

- Up — Moves the component up in the order in which it starts.
- Down — Moves the component down in the order in which it starts.
- Edit — Allows you to modify the settings for the component.
- Remove — Removes the component from the launch configuration
- Add — Allows you to add a correlator or an external process to the launch configuration. To add a component, click the Add button and select the type of component you want to add.
- Restore Default — Sets the launch configuration to use the default launch configuration.
- Connections — Displays the Connections dialog, which lets you add and remove connections between correlators. See "Connecting correlators" on page 113.

6. The Environment tab lists any additional environment variables needed to run any of the processes started by this launch configuration.



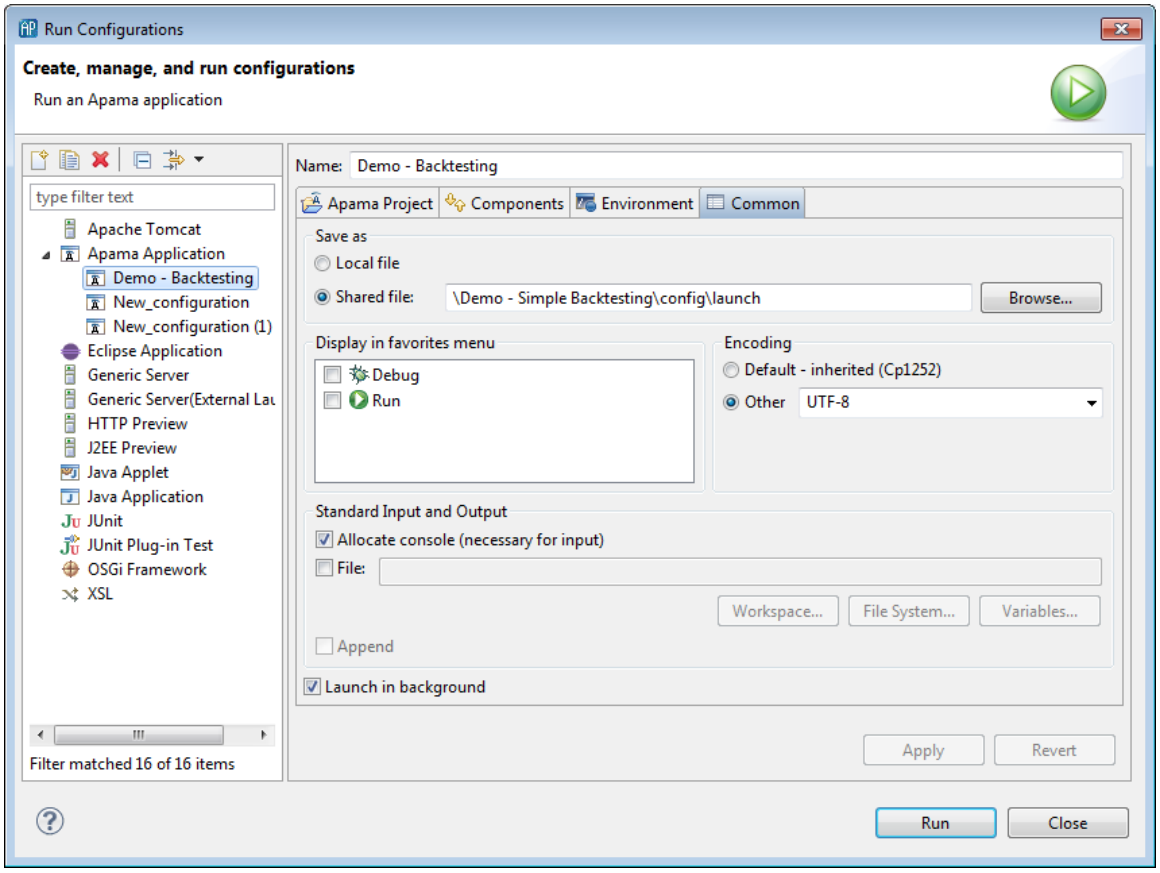
Then provide information here in the following ways:

- New —Specify a new environment variable.
- Select — Select an environment variable from the list of Eclipse environment variables.
- Edit —Modify the value of an environment variable.
- Remove — Remove an environment variable from this configuration.

Apama recommends that you append new environment settings to the native environment as otherwise it will be necessary to manually specify the standard Apama environment variables in order for the process to start correctly.

Note that to add a suffix or prefix to an existing environment variable, a new environment variable of that name should be created, and the existing value specified as part of the variable's value, for example, `PATH=${env_var:PATH};C:\my path`.

7. The Common tab specifies additional attributes for running this launch configuration.



This tab displays the following fields:

- Save as
  - Local file
  - Shared file — This is the default selection and the default path is the `config\launch` folder in your project directory. Click Browse to navigate to and select another location that is available to all users sharing this project.
- Display in favorites menu

  Select the appropriate checkbox to display this configuration as a choice in the drop-down menu of the Debug or Run button in the Apama Studio toolbar.
- Console encoding — The default encoding for output to the console is Cp1252. To encode console output in a different format, click Other and select the encoding you want. For example, UTF-8.
- Standard Input and Output

  Allocate Console (necessary for input) — For correlator launch configurations, there are no reasons not to allocate a console. Allocating a console does not affect performance.

  File — If you want to capture correlator output in a file, navigate to and select a file to contain the correlator output.

  Append — When you capture correlator output in a file, indicate whether you want to append the output to the specifies file.
- Launch in background — The default is that the correlator runs as a background process.

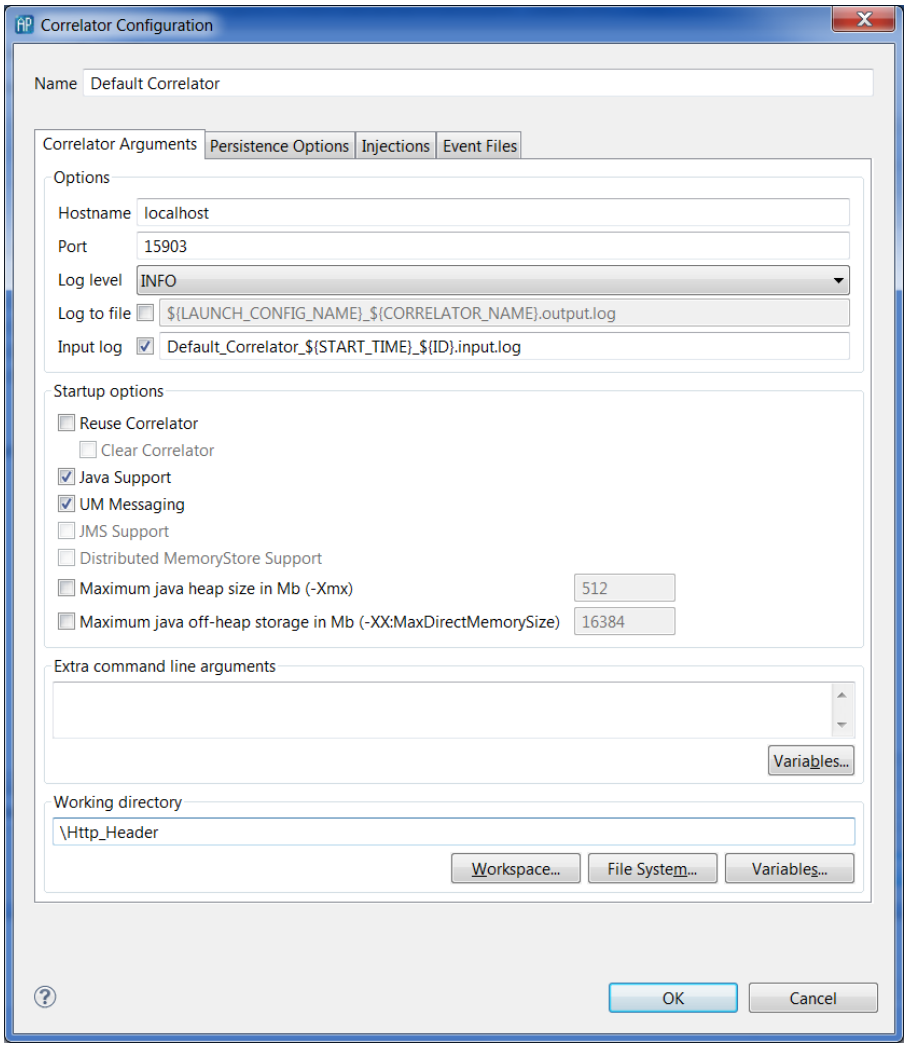
Adding a correlator

When you add a correlator to the launch configuration, Apama Studio displays the Correlator Configuration dialog, which includes these tabs:

- Correlator Arguments
- Persistence Options
- Injections
- Event Files

Correlator arguments

The Correlator Arguments tab specifies the options and arguments that Apama Studio uses to start the correlator.



You can accept the default values or change one or more arguments. You can also add arguments. In addition, you can also control the behavior with the following options:

- Hostname — `localhost`. The name of the correlator host machine.
- Port — This is the port on which the correlator listens for monitoring and management requests.
- Log level — Default is INFO.
- Log to file — Default is `${LAUNCH_CONFIG_NAME}_${CORRELATOR_NAME}.output.log`. This is the correlator status log.
- Input log — Default is `Correlator_1_${START_TIME}_${ID}.input.log`. This log contains all incoming messages. See *Using an input log to diagnose problems* in *Deploying and Managing Apama Applications*.
- Reuse Correlator — Select this option to use an already running correlator.
- Clear Correlator — Select this option to delete all the correlator contents when the launch starts.
- Java Support — Select this option to provide Java support for JMon applications. Normally this is automatically selected when a JMon application is created in Apama Studio.
- UM Messaging — Select this option to inject the project's `UM-config.properties` file, which is in the project's `config` folder. This is equivalent to specifying the `-UMconfig` option when starting a correlator. Uncheck this box if you want to run the project without using UM. If the project does not contain a UM configuration properties file, this option is disabled.

  See "Using Universal Messaging" in *Deploying and Managing Apama Applications*.
- JMS Support — Select this option to provide JMS support for correlator-integrated messaging. Normally this is automatically selected when the correlator-integrated messaging adapter for JMS is added to an Apama application.
- Distributed MemoryStore Support — Select this option to provide support for distributed MemoryStore. With the distributed MemoryStore, you can create distributed stores that provide access to data that will be shared among Apama applications running in separate correlators.
- Maximum java heap size in Mb (-Xmx). Default is 512.
- Maximum java off-heap storage in Mb (-XX:MaxDirectMemorySize). Default is 16384.
- Extra command line arguments. Specify any additional arguments for starting the correlator. See *Starting the event correlator* in *Deploying and Managing Apama Applications*.

Persistence options

The Persistence Options tab specifies the correlator persistence settings Apama Studio will use when it runs this launch configuration.

- Enable correlator persistence — When set, this tells the correlator to start with the persistence option.
- Directory location — This specifies the location on disk of the persistent store used to preserve correlator state.
- Startup options — These options specify the startup behavior of the correlator with respect to how it handles its persistent store.
  - Clear state and re-inject — Specifies that you want to clear the contents of the recovery datastore.

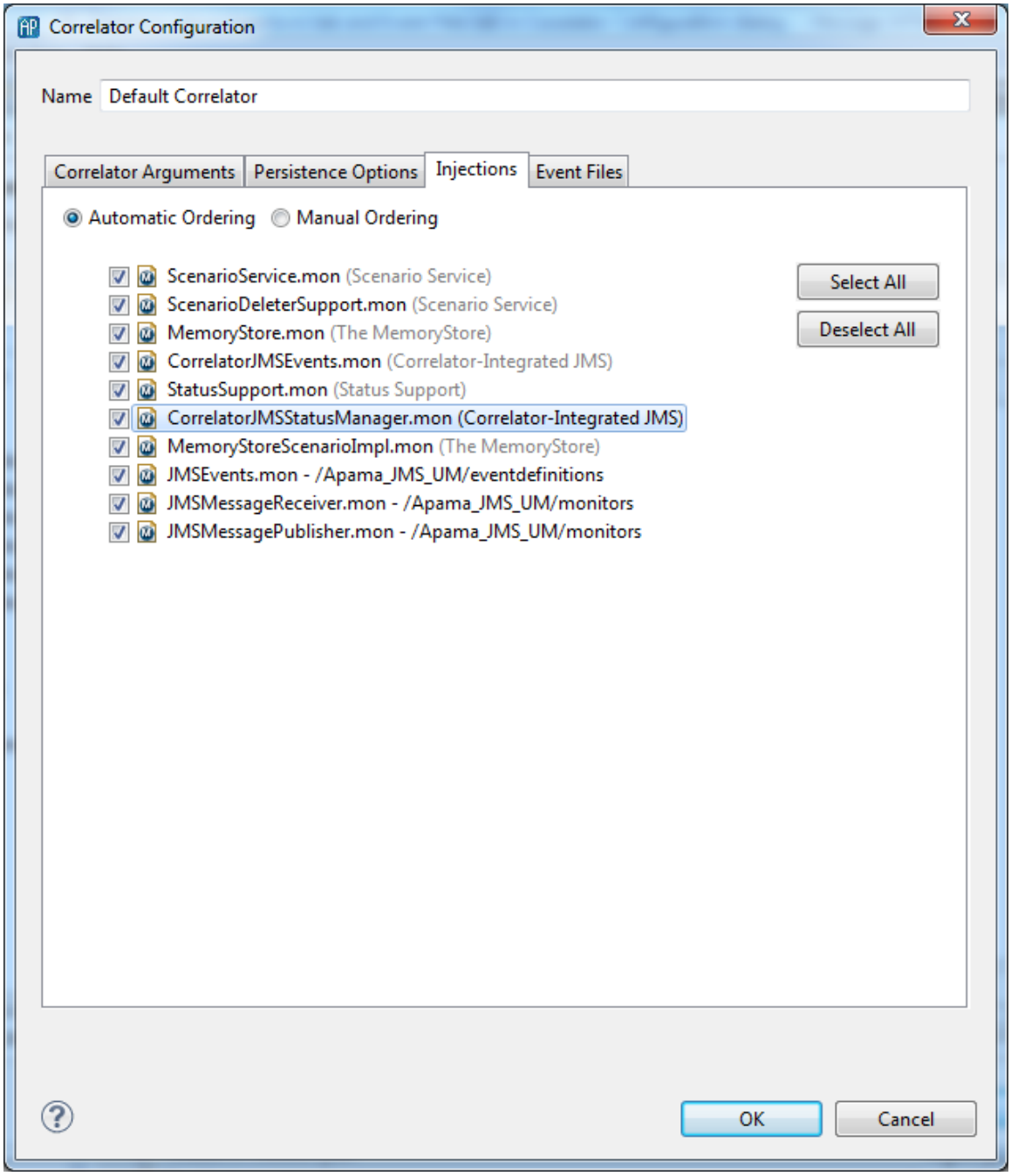
- Resume from persistent state — Specifies that the correlator will restart using the state from its persistent store as of the last committed snapshot.
- Prompt for startup options dialog during correlator launch — Specifies that Apama Studio will present a dialog when launching the application. The dialog asks whether to clear the correlator state information and re-inject the application code or to resume from the state information stored in the last committed snapshot.

For more information on correlator persistence, see "Using Correlator Persistence" in *Developing Apama Applications in EPL*.

For more information on persistence options for starting the correlator, see "Starting the event correlator" in *Deploying and Managing Apama Applications in EPL*.

Injections

The Injections tab lists the files, except event files, that will be injected or sent into the correlator when Apama Studio runs this launch configuration. This includes the files that are in your project as well as additional files that have been added to the project build path. See "Defining custom launch configurations" on page 103.



By default the Injections tab lists all EPL files in the project. The files are listed in dependency order. This is the order in which Apama Studio will load the files. Apama Studio determines the dependency order when it builds the project. In almost all projects, you do not need to change the order. However, if necessary, you use this tab to change the order in which Apama Studio injects the project's EPL files.

When Automatic Ordering is selected you have the option to exclude files from the launch configuration by unchecking them. When you run this project the unchecked files are not injected.

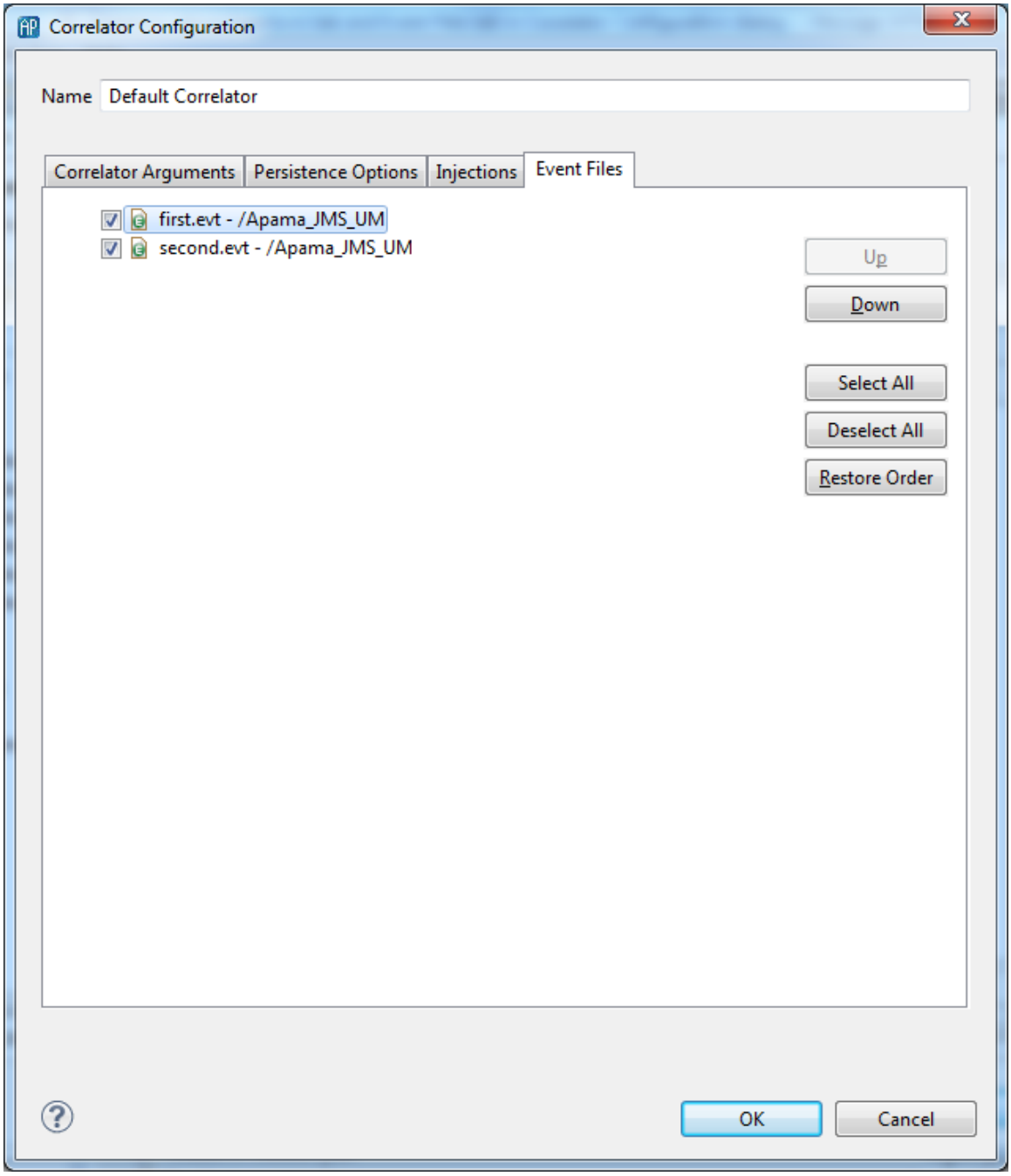
When you select Manual Ordering the Up and Down buttons appear. Select the file whose position you want to change and click the Up or Down button as many times as needed. When Manual Ordering is selected Apama Studio does not resolve any conflicts. It is up to you to correctly order the files.

If you uncheck a check box, the corresponding file remains in the list, but will not be loaded during this launch configuration.

All selected EPL files are injected and then all selected `.evt` files (on the Event Files tab) will be injected.

Event Files

The Event Files tab lists the event files that will be injected or sent into the correlator when Apama Studio runs this launch configuration. This includes event files that are in your project as well as additional event files that have been added to the project build path. See "Defining custom launch configurations" on page 103.



By default the Event Files tab lists all `.evt` files in the project. The files are listed in lexicographic order by file name. This is the order in which Apama Studio will inject the files.

If necessary, you use this tab to change the order in which Apama Studio injects the project's event files. Select the file whose position you want to change and click the Up or Down button as many times as needed. It is up to you to ensure that the files are injected in the correct order.

If you uncheck a check box, the corresponding event file remains in the list, but will not be injected during this launch configuration.

All selected EPL files (listed on the Injections tab) are injected and then all selected `.evt` files will be injected.

Connecting correlators

When you are creating a custom launch configuration you can specify connections between correlators. This is similar to connecting correlators with the `engine_connect` correlator utility.

Each connection you add goes in only one direction. For example, suppose you define a connection from `correlatorA` to `correlatorB`. This connection lets `correlatorA` send events to `correlatorB`. If you want `correlatorA` to receive events from `correlatorB` you must add a second connection that specifies `correlatorB` as the source of the connection and `correlatorA` as the target of the connection.

To connect correlators:

1. In the Run Configurations dialog, select the Components tab and ensure that the correlators you want to connect are listed as components. See "Adding a correlator" on page 108.
2. Click Connections.
3. In the Connections dialog, click Add, which activates a row in the connections table.
4. In the activated row, in the From field, select the name of the correlator that is the source of this connection. If the name of the correlator you want to connect does not appear, click Cancel to return to the Components tab and then select Add > Add Correlator.
5. In the To field, select the name of the correlator that is the target of this connection.
6. In the Channel field, specify one or more channels to be used by this connection. Use a comma as a separator. For example, `channelA, channelB`.
7. In the Flags field, specify any options that are supported by the `engine_connect` utility.

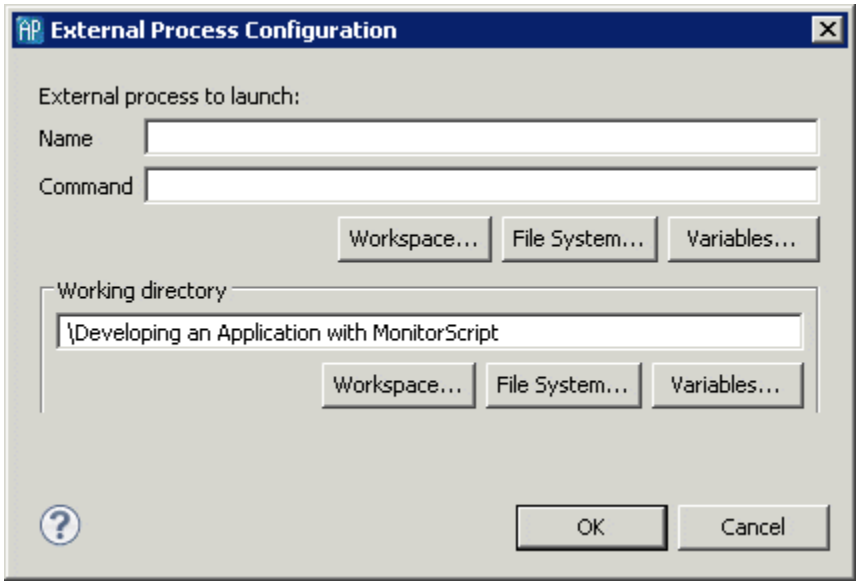
   See "Event correlator pipelining" in *Deploying and Managing Apama Applications*.

After adding the desired connections, click OK to return to the Components tab.

To remove a connection, select the row that defines the connection and click Remove.

Adding an external process

When you add an external process to the launch configuration, Apama Studio displays the **External Process Configuration** dialog.



This dialog allows the user to add and edit an external process associated with the launcher. You can enter the relevant information in the following fields:

- Name — The name is a unique name in the launcher.
- Command — This is the command line string that is used to invoke the external process. This command string can use the variables from the Workspace, File System, or Variables configured in Eclipse. These variables are replaced with their corresponding values when launched.

Testing a subset of your apama application

In a large Apama application, you might want to test a subset instead of the entire application. The best way to do this is to define a launch configuration that injects only the monitors you want to test. Use this configuration only to test the subset. Create a different launch configuration to test the entire application.

The reason you want a separate configuration for testing the subset is that you must disable the monitors in your project that are not in the subset you want to test.

To disable a file in a launch configuration:

1. Define the launch configuration as you normally would. See "Defining custom launch configurations" on page 103.
2. In the Create, manage, and run configurations dialog, on the Components tab, select the correlator in which you want to perform the testing and click Edit. Or, click Add > Add Correlator to add the correlator on which you want to do the testing.
3. In the Correlator Configuration dialog that appears, on the Initialization tab, deselect the file(s) you do not want to inject.
4. Click OK, Apply, and then Run or Close.

Monitoring apama applications

Apama Studio provides various ways to monitor a running Apama application.

- **Console** view
- **Engine Information** view
- **Engine Receive** view
- **Engine Status** view
- **Scenario Browser**
- Dashboards

Note that in the **Developer** perspective it is possible to launch multiple projects at any one time. By default, the **Console** view and the **Scenario Browser** view will display information about the most recently launched correlator. To monitor a different correlator, select Window > Show View > Other > Debug from the Apama Studio menu and select the **Debug** view — this lists all running launches. Select the correlator that you wish to monitor from this list.

Opening the **Apama Runtime** perspective (either from the perspective shortcut bar in the top right, or selecting Window > Open Perspective > Other > Apama Runtime from the Apama Studio menu) results in a convenient layout of all the available Apama Runtime views. This is the recommended way to make use of these views. You can switch back to the Apama **Developer** or **Workbench** perspective at any time.

Console view

The **Console** view displays information concerning a running Apama application. An application can have several consoles:

- Correlator — Displays output from the correlator.
- Engine Inject —Displays initialization information injected to the correlator.
- Engine Send — Displays information from Apama components such as dashboards that stream data to the correlator.
- Correlator Initialization — Displays information about the correlator initialization including the Java files, `.mon` files (monitors), and `.sdf` files (scenarios) that have been injected and the `.evt` files (events) that have been sent and whether the actions succeeded or failed.

To view one of these consoles, click the drop down arrow of the Display Selected Console button  and select the console you want.

Using the Engine Information view

The **Engine Information** view inspects a running correlator and displays defined contents of the correlator. It displays the same information as the Apama command line tool `engine_inspect`.

The information is grouped as follows (click on the right-facing triangle to view the contents of each group):

Aggregates — The names of custom (user-defined) aggregate functions that have been injected. You use aggregate functions in stream queries. Apama built-in aggregate functions are not listed.

Contexts — The names of any user-defined contexts, how many monitor instances are running in each context, how many entries are on each context's input queue, and the names of any channels each context is subscribed to.

Events — The names of all event types in the correlator, and the number of templates of each type, as defined in listener specifications.

Java Monitors — The names of all Java applications in the event correlator and the number of event listeners each Java application has created.

Monitors — The names of all monitors in the event correlator and the number of monitor instances each monitor has spawned.

Plugin Receivers— The names of any plug-in receivers, the channels each plug-in receiver is subscribed to, and the number of items on each plug-in receiver's input queue. A p;ug-in receiver is a correlator plug-in that is subscribed to one or more channels.

Receivers— The names of any external receivers, the address of each external receiver, the channels each external receiver is subscribed to, and the number of entries on the output queue of each external receiver.

Timers — Displays the current EPL timers active within the system. The four types of timers that might be displayed here are `wait`, `within`, `at`, and `stream`. The `stream` timers are those set up to support the operation of a stream network.

This view allows direct deletion of the defined named entities using the Delete button  on the tool bar or from the context (right-click) menu.

The user can send an event from this view as well using the send operation by clicking the Send button  on the toolbar or via the context menu. Double clicking an event item also invokes this operation.

Displaying information in the **Engine Information** view can slow performance, so you may want to close the view if Studio is not very responsive.

Using the Engine Receive view

The **Engine Receive** view shows all events generated from the connected correlator. Each batch of events is separated with an optional separator.

- Select All — Select all text in the console.
- Clear — Clear all text in the console.
- Copy — Copy all selected text to the clipboard

- Show Separator — Show a separator line in each event batch.
- Show Date/Time of batch — Show a date/time of each event batch

You can also change whether to display the separator line and change how it is displayed by clicking the drop down arrow at the top right of the view to display the menu and select Options. See "Engine Receive Viewer preferences" on page 117.

Click the Toggle Connection button  to temporarily disconnect the **Engine Receive** view from the correlator.

When the Toggle Connection button is pressed, the console will not update any further events from correlator. The background of the console will be changed to indicate the temporary disconnect mode. The Select All, Clear, and Copy actions will still work in this mode.

To resume from the temporary disconnect mode, simply click on the Toggle Connection button again to resume the connection. The console will be cleared and new events will be shown in the console again.

Engine Receive Viewer preferences

You can change the display options of the **Engine Receive** view by clicking the drop down arrow at the top right of the view. From the menu select Options, which displays the**Engine Receive Viewer Preferences** dialog. This dialog contains settings that specify how information is displayed in the **Engine Receive** view.

Display a separator — Toggles whether each batch of events is visually separated.

Display date/time for batch — Toggles whether the date and time for each batch of events is displayed.

Custom text for separator — Specifies the characters composing the separator.

Max window size (KB) — Select a setting between 100 and 10000.

Using the Engine Status view

The **Engine Status** view displays the information about the correlator status. The information is the same as the output of Apama command line tool `engine_watch`.

Uptime(ms) — The time in milliseconds since this correlator was started. This figure is unaffected if the state of the correlator is restored from a checkpoint file.

Number of contexts — The number of contexts in the correlator. This includes the main context plus any created contexts.

Number of monitors — The number of monitor definitions injected into the correlator. This figure changes upwards and downwards as monitors are injected, deleted or just expire. A monitor expires when each of its instances dies, has no listeners left, or causes a runtime error.

Number of sub-monitors — The number of monitor instances across all contexts in the correlator. In monitors, spawn actions create monitor instances. This figure changes upwards and downwards as monitor instances are spawned, killed or just expire.

Number of java applications — The number of JMon applications loaded in the correlator. JMon applications do not expire, so this value only decreases when you explicitly unload a JMon application.

Number of listeners — The number of event listeners created by monitor instances across all contexts, plus the number of JMon applications.

Number of sub-listeners — The number of sub-listeners that have been created by listeners across all contexts.

Number of event types — The total number of event types defined within the correlator. This figure decreases when you delete event types from the correlator.

Events on input queue — The sum of the number of events on the input queue of each context. The main context has its own input queue and any user-defined contexts each have an input queue.

Events received — The total number of events ever received by the correlator. A checkpoint preserves this value. If you restore the state of the correlator from a checkpoint file, this number reflects the total number of the events seen by the correlator from which the checkpoint was originally made. Note that if an event is on an input queue, it has been received but not processed.

Events processed — The total number of events processed by the correlator in all contexts. This includes external events and events routed to contexts by monitors. An event is considered to have been processed when all listeners that were waiting for it have been triggered, or when it has been determined that there are no listeners for the event.

Events on internal queue — The sum of the number of routed events on the input queue of each context. The internal routing queue in each context is a high priority queue for events that you internally routed with the `route` instruction in your EPL files. The correlator always processes routed events before processing events on the input queue.

Events routed internally — The sum of the number of events routed in each context since the correlator started. A checkpoint preserves this value. If you restore the state of a correlator from a checkpoint file, this number reflects the total number of the events routed to the internal queues for the correlator from which the checkpoint was originally made.

Number of consumers — The number of event consumers registered with the correlator. Event consumers receive events emitted by the correlator.

Events on output queue — The number of events waiting on the correlator's output queue to be dispatched to any registered event consumers.

Output events created — The total number of output events created by the correlator. A checkpoint preserves this value. If you restore the state of a correlator from a checkpoint file, this number reflects the total number of output events created by the correlator from which the checkpoint was originally made.

Output events sent — The total number of output events that the correlator has sent to event consumers. For example, suppose the correlator created 10 output events and sent each event to two consumers. The number of output events sent is 20. A checkpoint preserves this value. If you restore the state of a correlator from a checkpoint file, this number reflects the total number of output events sent by the correlator from which the checkpoint was originally made.

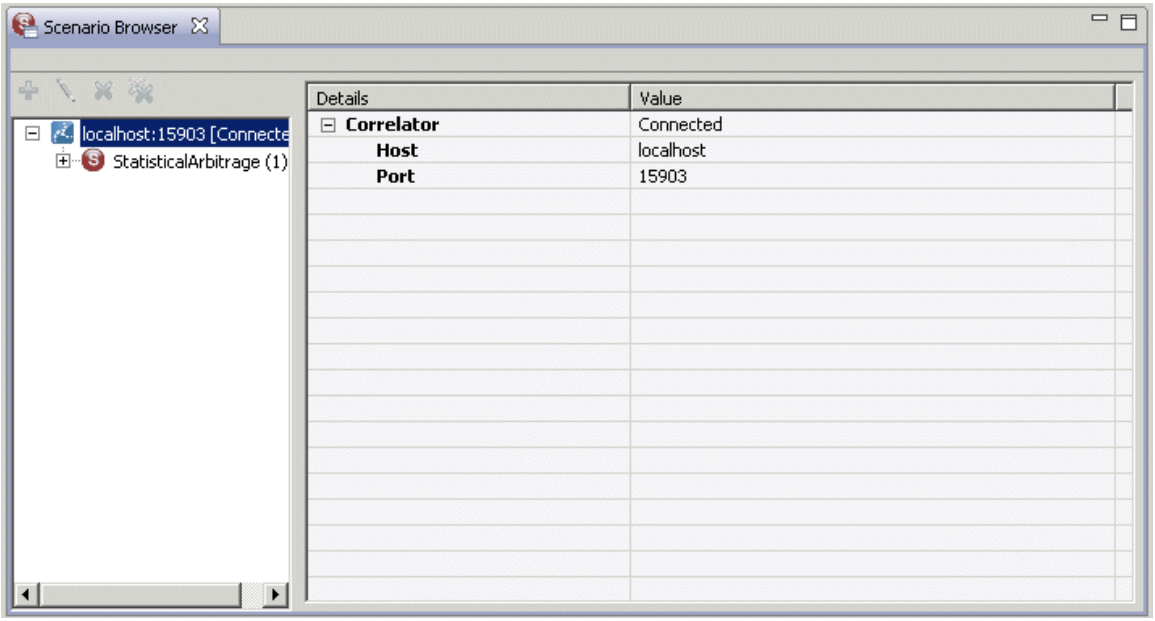
Using the Scenario Browser view

Apama Studio's **Scenario Browser** view displays the scenario definitions loaded into the correlator along with the metadata for those scenarios. The Scenario Browser also displays scenario instances that are running along with the scenario output values. In the Scenario Browser you can add, edit, and delete scenario instances.

Displaying the Scenario Browser

By default, in the **Workbench** perspective, the Scenario Browser is opened when you launch an application. In the **Developer** perspective you need to display the **Scenario Browser** view manually by selecting Window > Show View > Scenario Browser from the Apama Studio menu. **Developer** perspective users may find it more convenient to switch to the **Runtime** perspective, either from the perspective shortcut bar in the top right, or by selecting Window > Open Perspective > Other > Apama Runtime. You can switch back to the **Apama Developer** perspective at any time.

The **Scenario Browser** view looks something like the following, showing running correctors in the left pane.

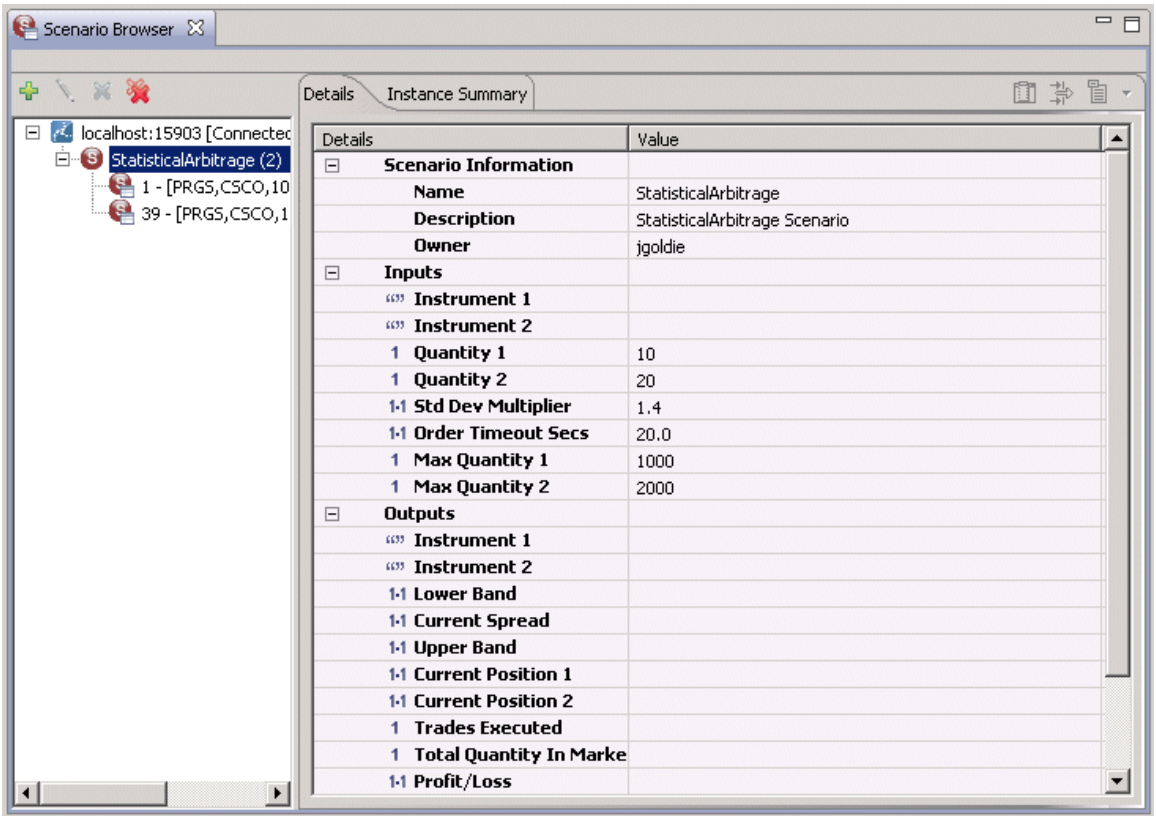


The details of the selected correlator are shown in the right pane.

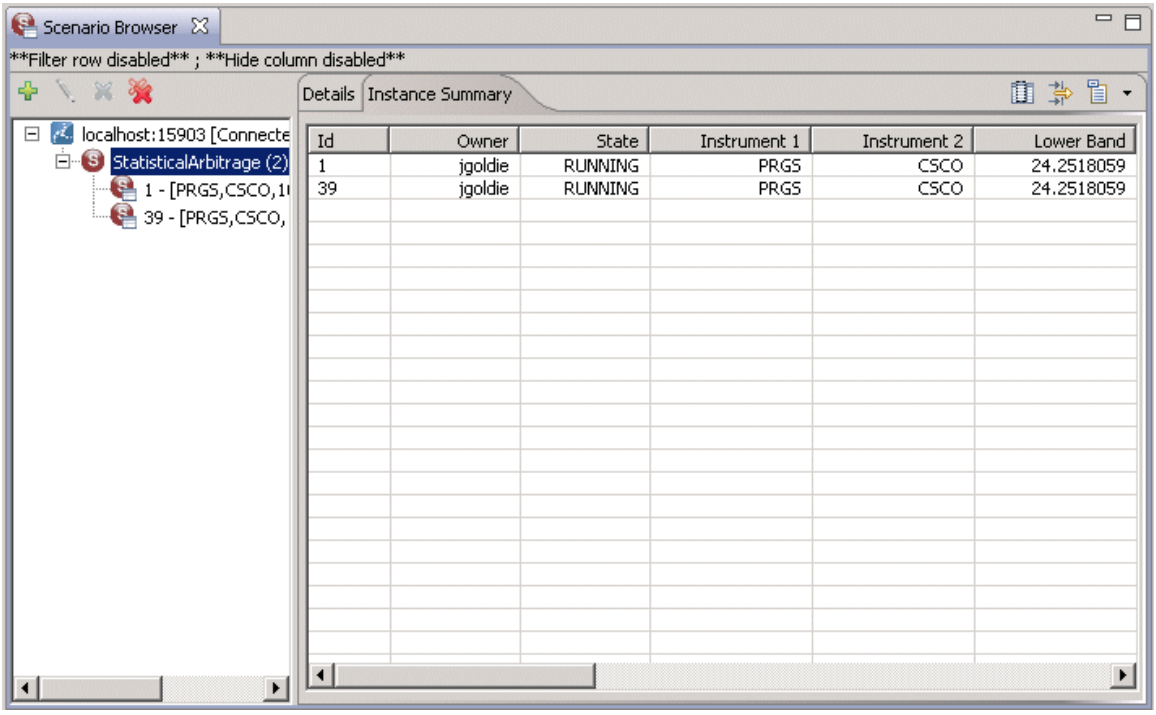
Browsing scenarios

To examine a scenario definition, click its name in the left pane. The illustration below shows the `StatisticalArbitrage` scenario definition. The Scenario Browser displays a list of the scenario's details in two tabs in the right pane: the Details tab and the Instance Summary tab.

The Details tab shows the metadata of the selected scenario; in the following illustration, the metadata for the StatisticalArbitrage scenario is displayed.



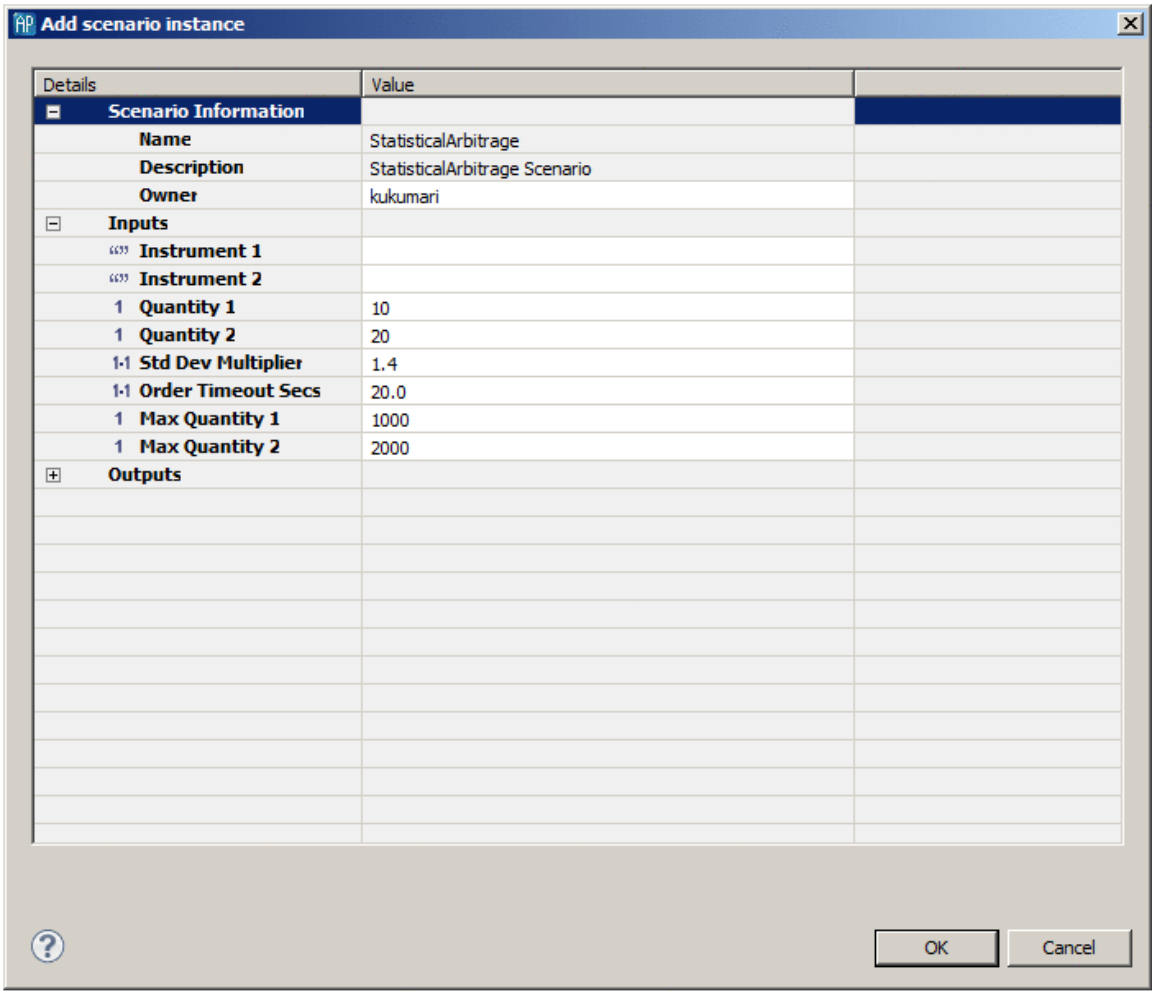
The Instance Summary tab shows the output fields of the selected scenario's instances. Each column represents a scenario output field. The following illustration shows the Instance Summary tab for the StatisticalArbitrage scenario with two scenario instances.



Creating new instances of scenarios

In the **Scenario Browser** view, you can create new instances of scenarios that are running in the application.

1. In the Scenario Browser's left pane, select the scenario for which you want to make a new instance.
2. Click the Add Scenario Instance button  near the top left of the Scenario Browser tab. The **Add Scenario Instance** dialog appears, showing the scenario's details including the input and output variables that make up the scenario definition in the left-hand column. The right-hand column displays the values of those variables.



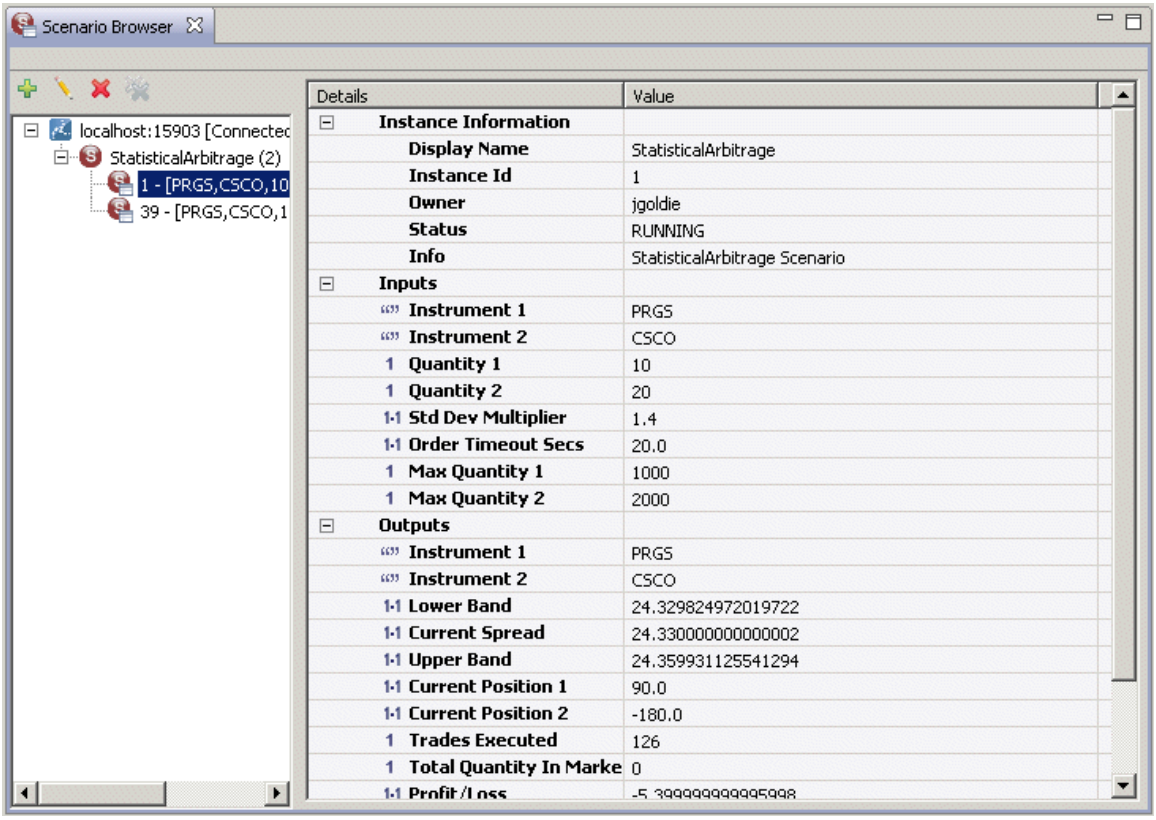
3. To create an instance of the scenario you specify values for the input fields in the right-hand column. Fields with default values are already filled in. You can add or edit values in cells that have a white background. To specify a value, double click in the empty cell and enter the information.
4. Click OK. Apama Studio adds the scenario instance to the Scenario Browser's left pane.

Viewing Scenario instances

You can display the details of a scenario instance in the **Scenario Browser** view as follows:

1. In the Scenario Browser, select the scenario ID of the instance you want. In the right pane, Apama Studio displays the names of the scenario variables in the left column and the current values of the variables in the

right column. The values of the output variables are continuously updated. The following illustration shows an example from the Statistical Arbitrage demo.

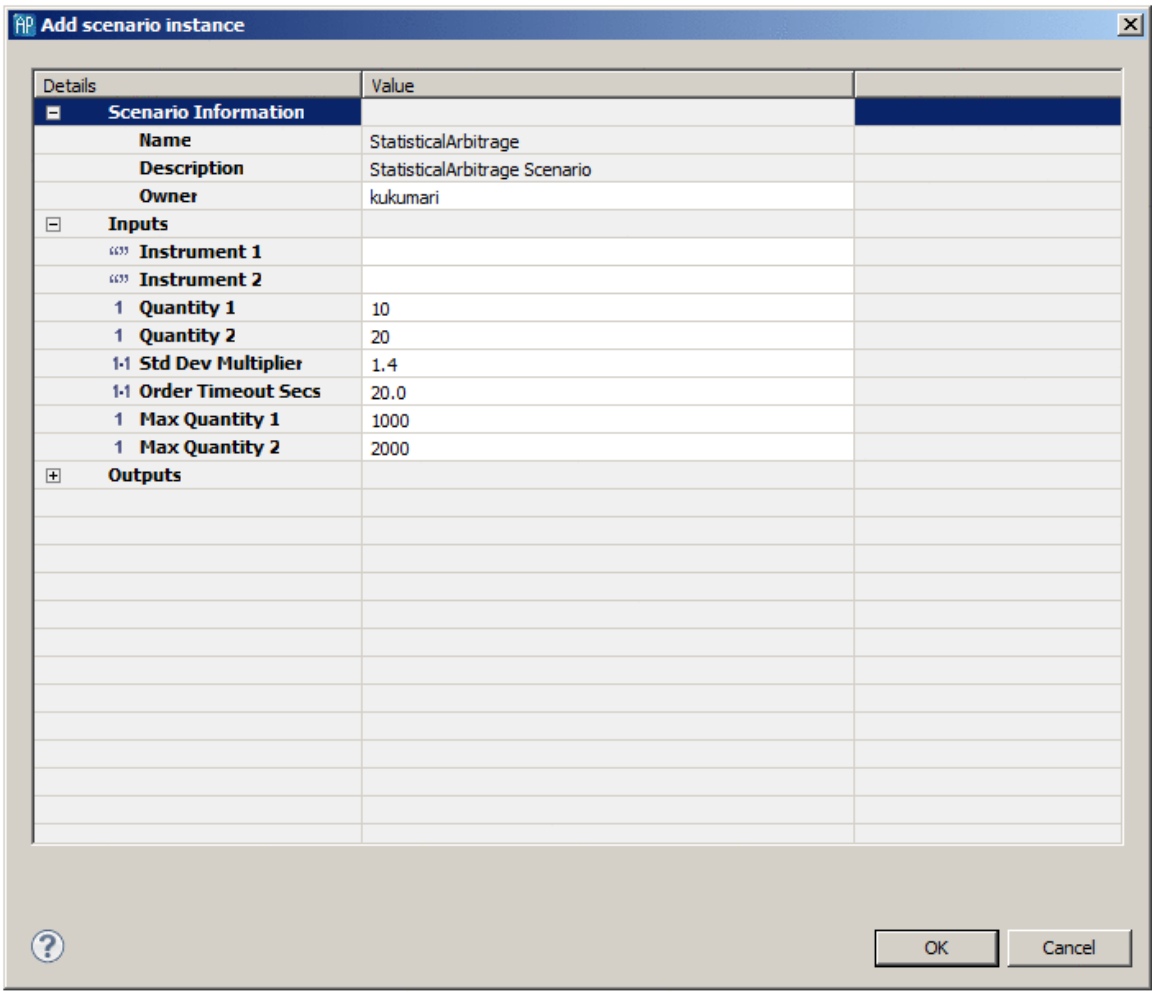


2. You can expand and collapse the display in the right pane by clicking the plus and minus symbols.

Editing a scenario instance

To edit the input values of a scenario instance:

1. In the left pane of the Scenario Browser, select the instance ID of the scenario instance you want to edit.
2. Click the Edit button  near the top left of the Scenario Browser. The **Edit scenario instance** dialog is displayed. While you are editing the values in the scenario instance Apama Studio suspends the display of output information.



3. Edit the values for the input fields in the right-hand column by double clicking in the desired cell and entering the new value.
4. Click OK. Apama Studio resumes displaying output values.

Deleting a scenario instance

To delete a scenario instance:

1. In the left pane of the Scenario Browser, select the instance ID of the scenario instance you want to delete.
2. Click the Delete button near the top left of the Scenario Browser.

Deleting all scenario instances

To delete all instances of a given scenario:

1. In the left pane of the Scenario Browser, select the name scenario for which you want to delete all the instances.
2. Click the Delete All button near the top left of the Scenario Browser.

Dashboards

When you launch a project that contains a dashboard, by default Apama Studio automatically starts the Dashboard Viewer and displays the project's default dashboard. If desired you can change this behavior as described in "Defining custom launch configurations" on page 103. You can specify that you do not want Dashboard Viewer to start automatically or that Viewer should display a dashboard other than the project's default dashboard. When you define a launch configuration you can also specify any command line options to need to be passed to the Dashboard Viewer.

Chapter 5: Debugging EPL Applications

You can debug Apama applications written in EPL in Apama Studio. As is the case when debugging other applications in Eclipse, when you debug EPL applications, you can set breakpoints to suspend the application at specific points, examine the contents of variables, and step through the application.

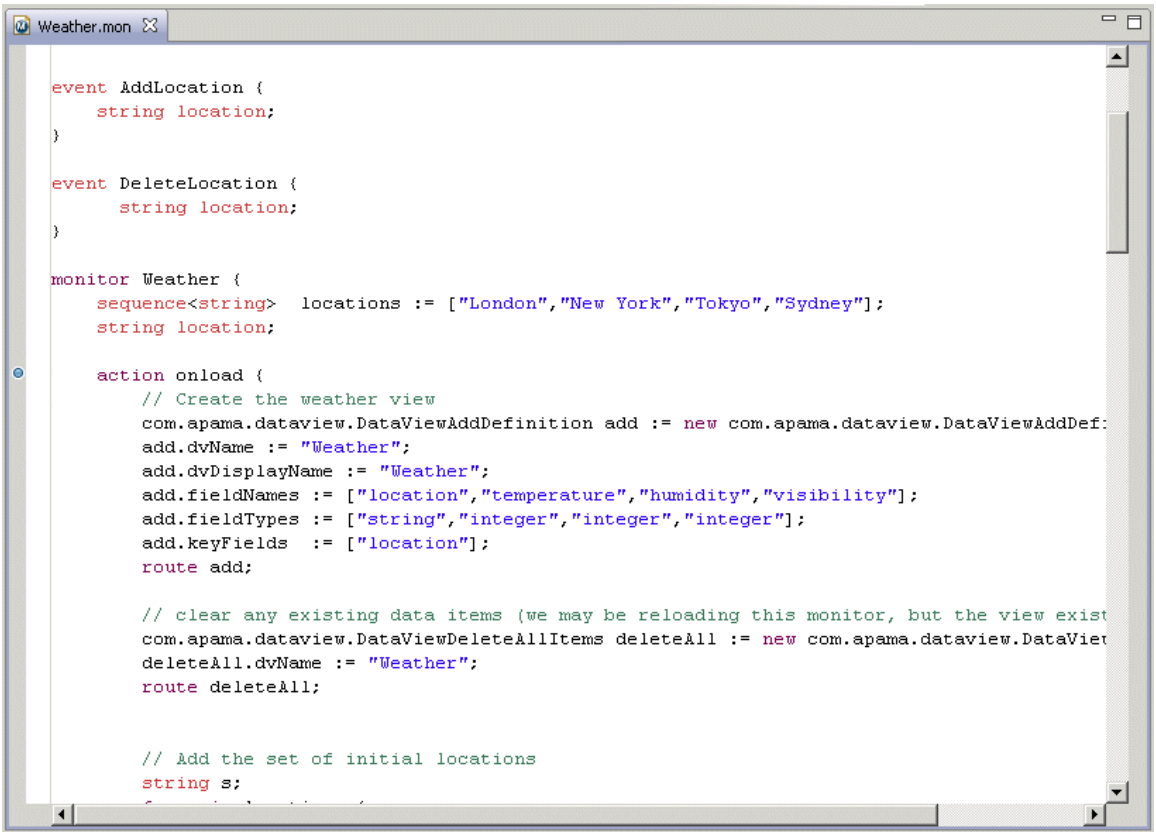
The basic tasks involved in debugging an EPL application in Apama Studio are:

1. Set breakpoints in the **Developer** perspective.
2. Create a debug launch configuration for the project.
3. Review the executing application in the **Debug** perspective, examining information such as the call stack, context status, and variable values.

Adding breakpoints

You can add breakpoints to any of the EPL files included in the project. To add a breakpoint to an EPL file:

1. In the **Developer** perspective open the EPL file in which you want to add a breakpoint.
2. Double click in the left margin of the desired line in the EPL file or right click in the left margin and select Toggle Breakpoint from the pop-up menu. The breakpoint is indicated by a solid blue circle in the left margin, as shown in the `action onload` line in the illustration below.



3. Repeat Step 1 and Step 2 for each breakpoint you want to set.

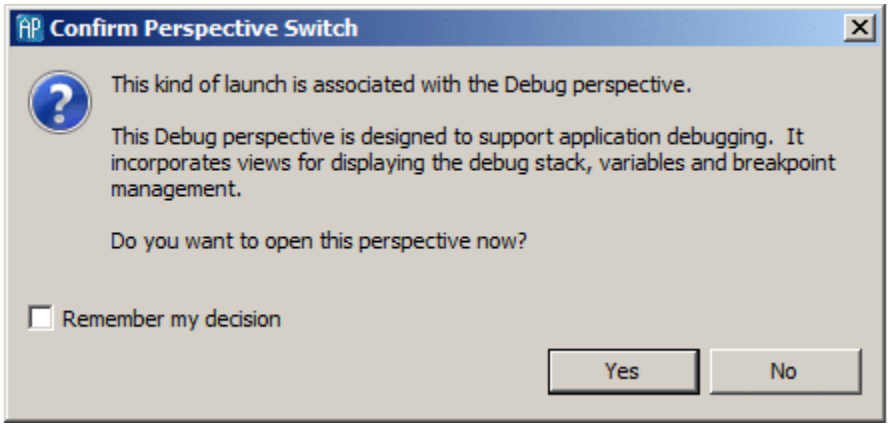
Note, when debugging an application on a remote machine, breakpoints are supported only inside EPL actions of monitors and events. Breakpoints inside listeners of actions or breakpoints outside of actions are not supported. See "Debugging a remote application" on page 129.

Debugging EPL Applications

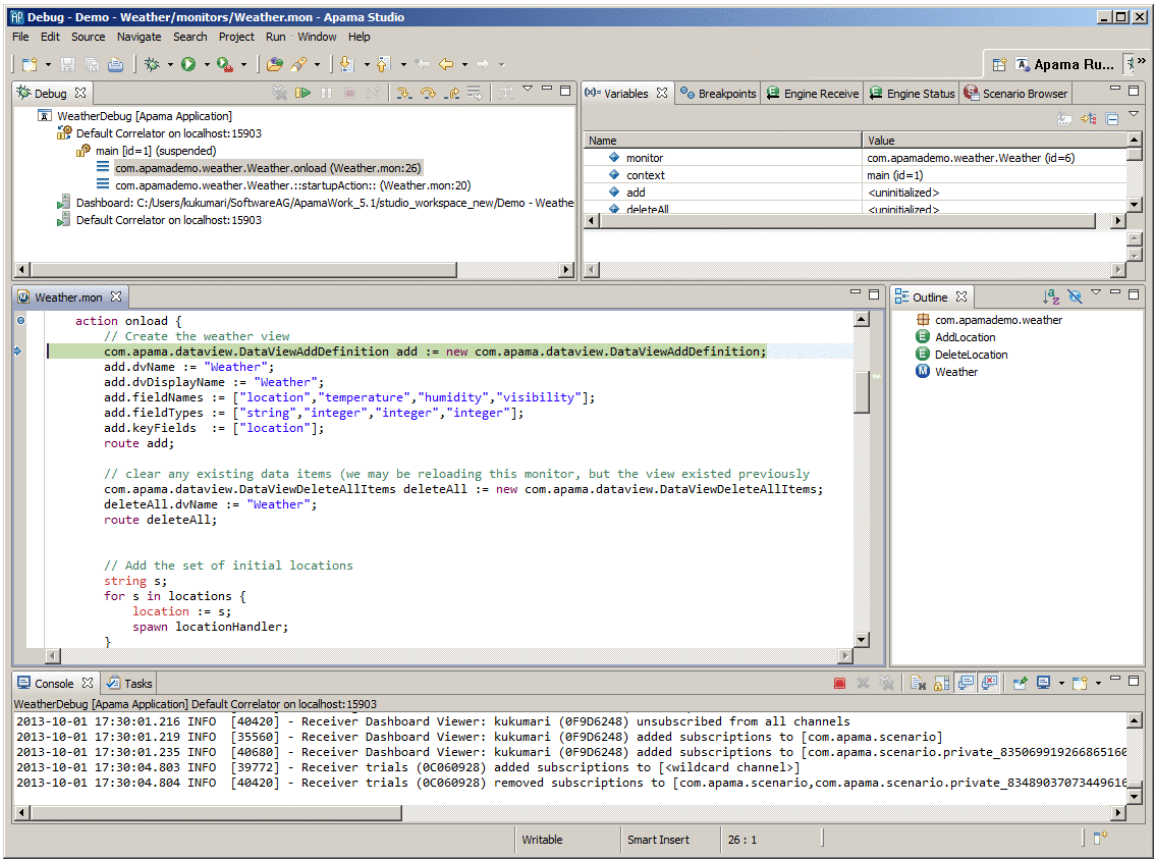
Launching a debug session

If a Debug Configuration has been defined for the Apama project, start a debugging session as follows. (If you need to create a Debug Configuration, see "Creating a debug configuration" on page 127)

1. From the Apama Studio menu, select Run > Debug Configurations.
2. From the **Debug Configuration** dialog, select the desired Debug Configuration and click Debug. The **Confirm Perspective Switch** dialog opens



3. Click Yes to display the **Debug** Perspective (and add a check to the Remember my decision check box if desired). The **Debug** perspective is opened.



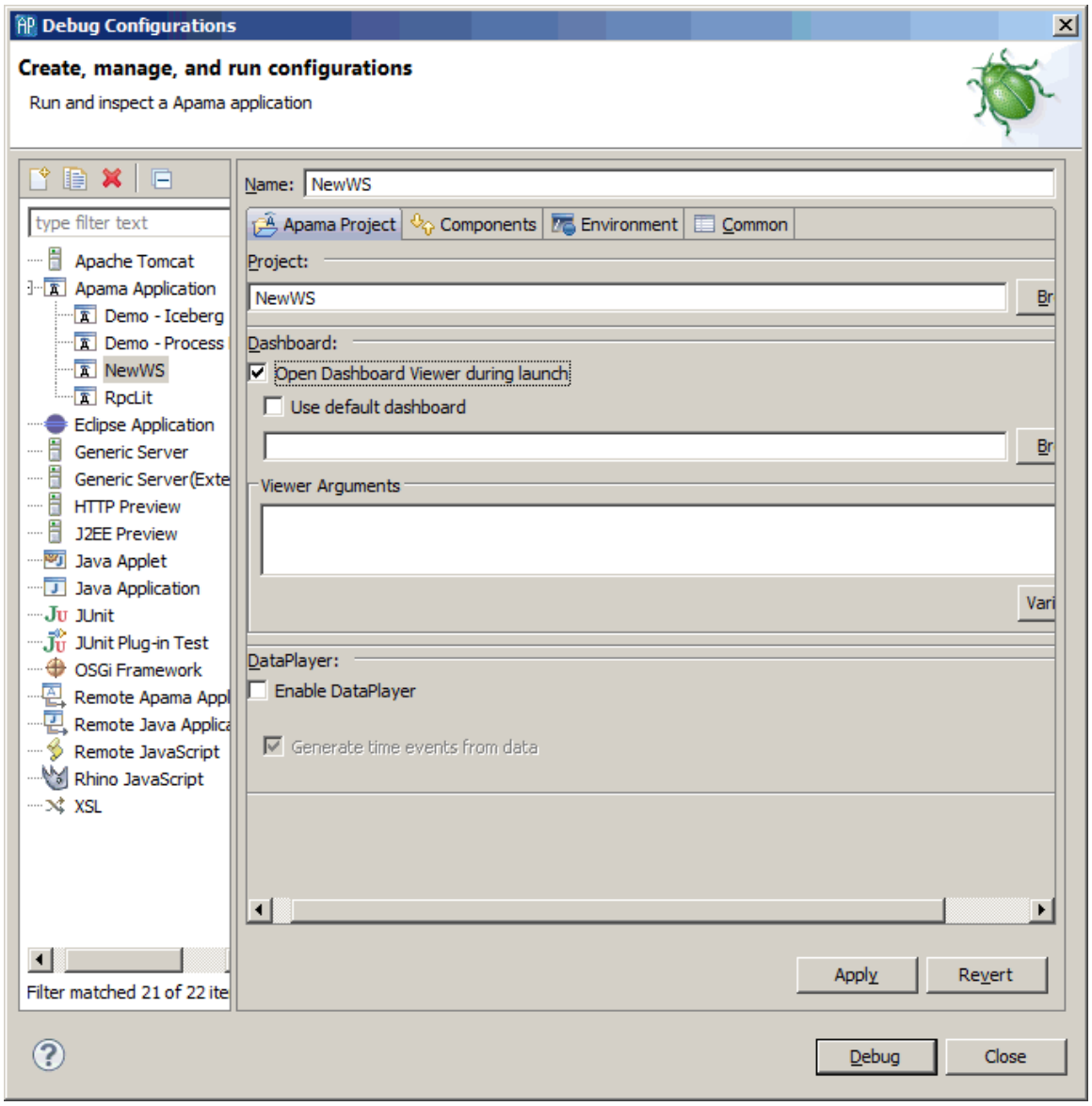
When debugging an EPL application, the **Debug** perspective is similar in operation to the standard Eclipse **Debug** perspective and includes the **Debug**, **Breakpoints**, and **Variables** views.

Debugging EPL Applications

Creating a debug configuration

If you need to create a new Debug Configuration:

1. From the Apama Studio menu, select Run > Debug Configurations. The **Profile configurations** wizard starts.



2. In the left pane, select Apama Application and click the New launch configuration button ( ).
3. In the right pane, on the Apama Project tab, specify the Name and the Project.
4. If you are going to debug an application running on a remote correlator:
   a. Select the Components tab.
   b. Click Add and select Add Correlator.
   c. In the **Correlator Configuration** wizard, on the Correlator Arguments tab, specify the Host, Port, and other Options.

      When creating a debug launch configuration, you do not need to specify the `-g` option in the Correlator Arguments tab. When you select Run > Debug , Apama Studio automatically starts the correlator with the `-g` option, which disables optimizations that hinder debugging. However, if you use Apama Studio to debug in a remote correlator, you must ensure that the

remote correlator was started with the `-g` option. You cannot debug in a correlator that was started without specification of the `-g` option. See "Deploy options" in "Starting the Event Correlator" in *Deploying and Managing Apama Applications*.

d. If necessary, specify any initialization options on the Initialization tab.

e. Click OK.

5. Click Debug. The **Confirm Perspective Switch** dialog opens.

6. Click Yes. Apama Studio switches to the **Debug** perspective.

Launching a debug session

Debugging a remote application

In Apama Studio you can debug an Apama application running in a correlator on a remote machine. The correlator on the remote machine must be started with the `-g` option, which disables optimizations that interfere with debugging.

To debug an application on a remote machine:

1. Start the application on the remote machine.

2. In Apama Studio on the local machine where you will do the debugging, select Run > Debug Configurations. This displays the **Debug Configurations** wizard.

3. In the **Debug Configurations** wizard, in the left pane select `Remote Apama Application` and click the New launch configuration button ().

4. In the right pane, on the Apama Project tab:

   a. Specify the Name of the debug configuration and the Project on the local machine where the debugging session is being run. The application in the project should match the application that is being debugged.

   b. Specify the Host and the Port of the correlator running on the remote machine.

   c. If desired, click Test Connection to verify the connection to the correlator on the remote machine can be made.

5. If desired, on the Source tab, add any needed resources. Generally, the project on the local machine would contain all the necessary resources.

6. Click Debug. The **Confirm Perspective Switch** dialog opens.

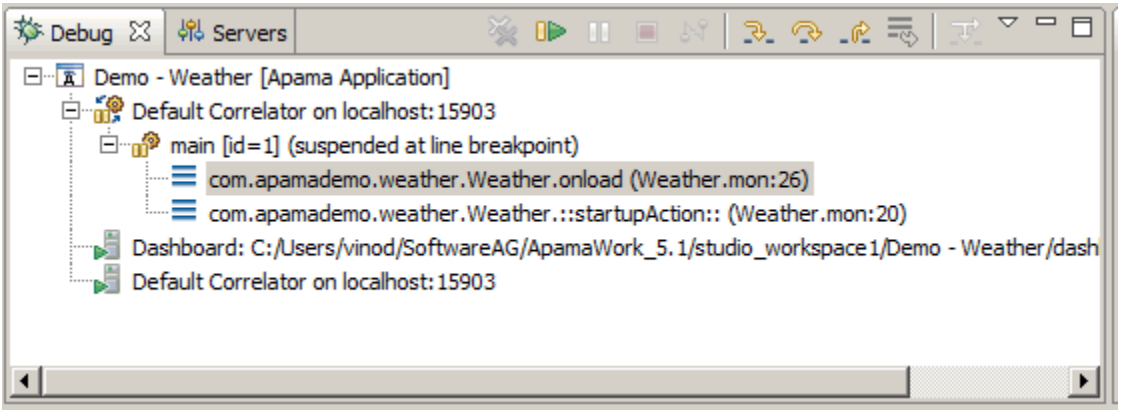
7. Click Yes. Apama Studio switches to the **Debug** perspective.

Note, when debugging a remote application, breakpoints are supported only inside actions of monitors and events. Breakpoints inside listener actions or outside of actions are not supported.

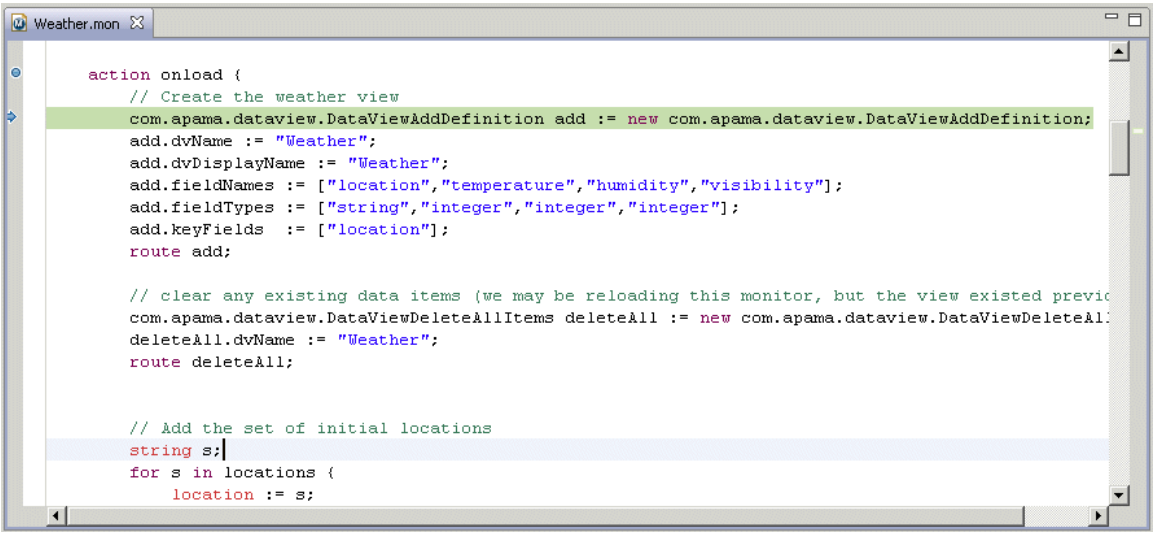
Debugging EPL Applications

Debug view

The **Debug** view shows the call stack and status of current contexts in the Apama application. When execution reaches a breakpoint, Apama Studio highlights the current context in the **Debug** view.



Also, when a breakpoint is reached, the EPL editor indicates the breakpoint in the code with an arrow in the left margin. The line of code that will be executed next is highlighted.



When a stack frame is highlighted in the **Debug** view, the **Variable** view displays the names and values of the associated variables, see "Variables view" on page 132.

The tool bar for the **Debug** view contains the following buttons for controlling execution:

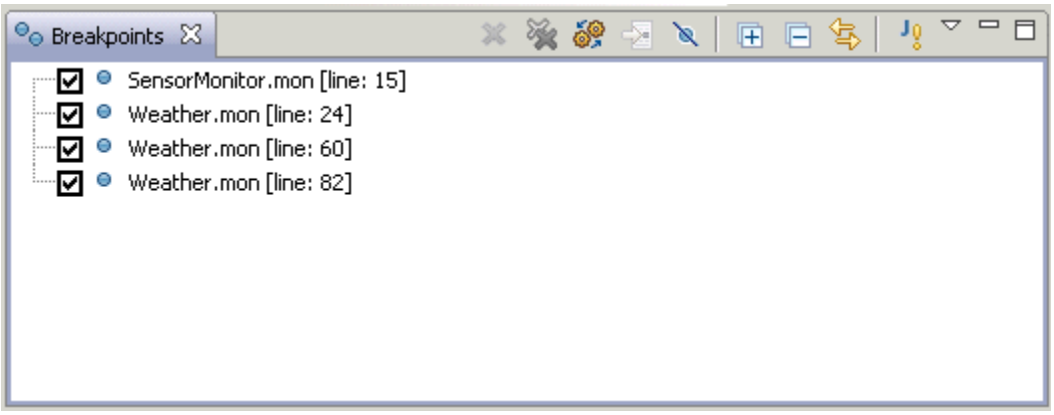
- Remove All Terminated Launches — Clears all terminated debug targets from the **Debug** view display.
- Resume — Resumes a suspended debug target.
- Suspend— Halts the execution of currently selected target.
- Terminate — Terminate the selected debug target.
- Step Into — Execute the current line and proceed to the next statement.

- Step Over — Steps over the highlighted statement and continues executing at the next line either in the same method or (if at the end of a method) continues in the method from which the current method was called.
- Step Return — Steps out of the current method. This stops execution after exiting the current method.

Debugging EPL Applications

Breakpoints view

The **Breakpoints** view lists the breakpoints you have set in the project.Double clicking a breakpoint displays the line in the EPL file that contains the breakpoint in the EPL editor. In the **Breakpoints** view, you can enable and disable the listed breakpoints.



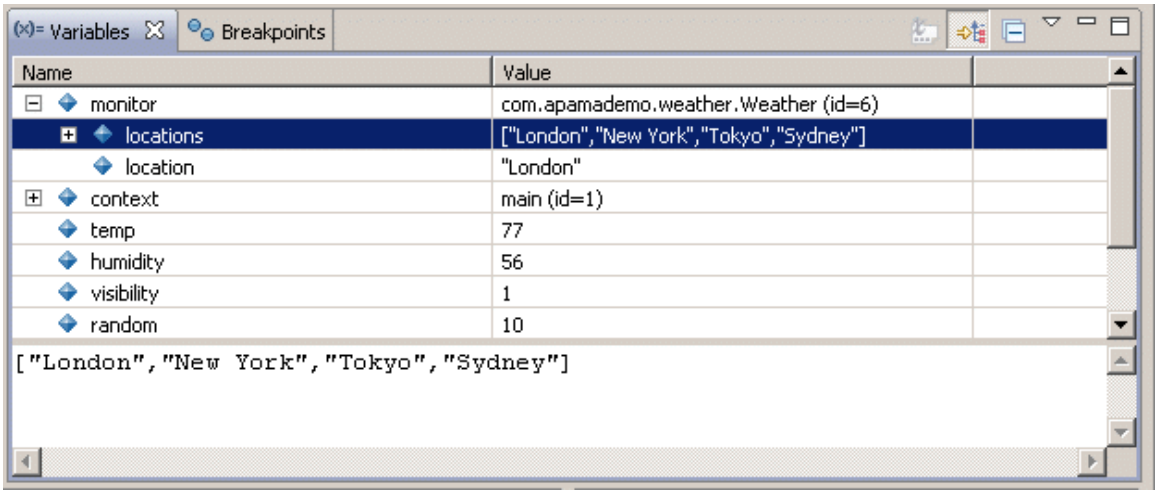
The tool bar for the **Breakpoints** view contains the following buttons for controlling execution:

- Remove Selected Breakpoints — Remove the selected breakpoint from the debug session.
- Remove All Breakpoints — Remove all breakpoints from the debug session.
- Show Breakpoints Supported by Selected Targets — Show breakpoints for selected target.
- Go to File for Breakpoint — Open file containing the selected breakpoint.
- Skip All Breakpoints — Skip all breakpoints.
- Expand All — Expand all of the items in the **Breakpoints** view.
- Collapse All — Collapse the display of all items in the **Breakpoints** view.
- Link with Debug view — Toggles whether or not the **Breakpoints** view is automatically updated when the **Debug** view changes.

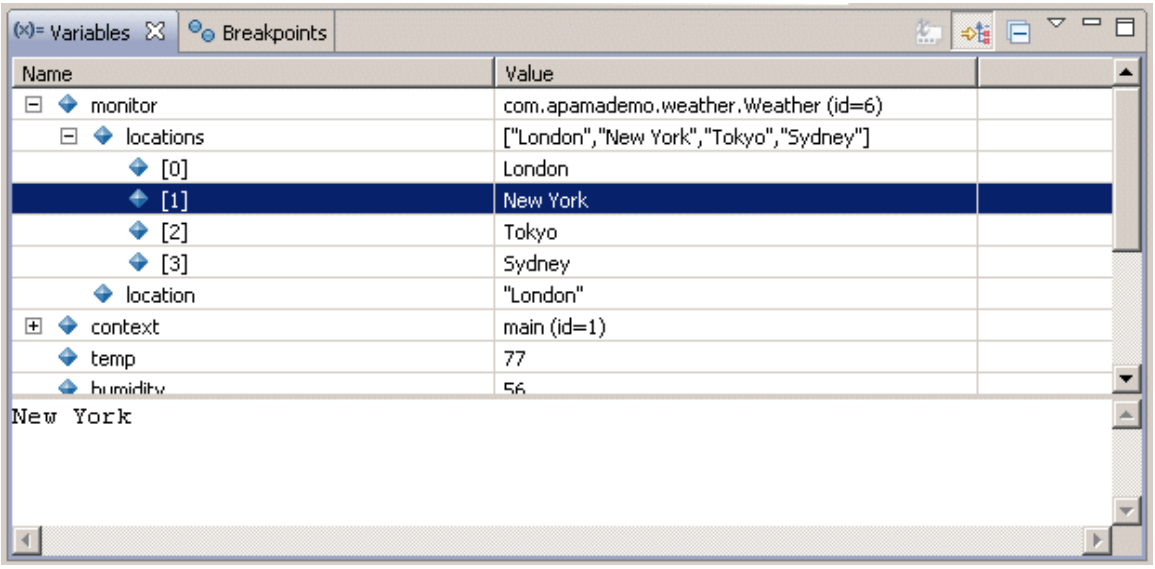
Debugging EPL Applications

Variables view

The **Variables** view displays information about the variables associated with the stack frame that is currently selected in the **Debug** view. The variables can be expanded to show their fields. The detail pane the area at the bottom of the view displays text. For a variable of a primitive type, this is the value of the object; for a complex variable, such as a sequence, dictionary, or event, it is the text representation of the object as returned by the object's `toString()` method.



For complex variables, you can click the Show Logical Structure button ( )to expand the variables to show their sub-values.



In the **Variables** view, you can change the layout of the display. Click the Drop-Down Menu button ( ) to the right side of the **Variables** view title bar and select Layout from the menu.

Debugging EPL Applications

Command-line debugger

You can also use Apama's `engine_debug` tool to control execution of your EPL code in the correlator and inspect correlator state. This tool is a correlator client that runs a single command from the command line. It is not an interactive command-line debugger. The interactive debugger in Apama Studio is more useful during the development of an application. In general, the command-line `engine_debug` tool is expected to be most useful during or after deployment of your application.

For more information on `engine_debug`, see "Using the command line debugger" in *Deploying and Managing Apama Applications*.

Debugging EPL Applications

Chapter 6: Debugging JMon Applications

This section describes practical information for developing and debugging JMon applications with Apama Studio. General knowledge of Java and Apama application development is assumed.

Apama Studio is built on the Eclipse IDE framework and as such running and debugging JMon applications for the Apama correlator engine is no more different than with any standard Java application. There are however a few things to consider:

- Single thread — For JMon applications, the correlator uses a single thread of execution. When programming in Java consider the importance of maintaining determinism by not routing, emitting, or enqueueing an event in a thread other than the current thread. Java provides no guarantee on execution order for separate threads, so try to keep your JMon applications single threaded at all times. Although multiple threads are currently allowed for JMon applications, this might change in the future, with the correlator JVM issuing a warning or possibly an error.
- Event definitions — To instantiate event definitions in your JMon application, the `Event` subclass in question needs to be defined and included in your `JAR` file. This is because the correlator uses a separate classloader for each application (that is, each JMon `JAR` file injected) and hence cannot share the event definitions across separate JMon applications. Also, a JMon application cannot make use of any event definition already present in the correlator. Any event definition (either a subclass of the JMon `Event` class or a definition in an EPL file) must be replicated for each JMon application and for your injected EPL files.

Preparing the correlator for remote debugging

Launching the correlator with the `-j` option enables the Java Virtual Machine (JVM) as a subprocess in correlator execution. When launching an Apama project in Apama Studio, specification of the `-j` option launches the correlator and launches its JVM as an external process. Consequently, you must enable a socket connection to the JVM so that you can connect the Apama Studio debugger to it. This also applies to launching a correlator through any other means.

Enable debugging in the correlator JVM by specifying the Java options for the correlator component in the launch configuration. The Java Debug Wire Protocol (JDWP) is the protocol used for communication between a debugger and the Java virtual machine (VM) that it debugs. Oracle's VM implementations require command line options to load the JDWP agent for debugging. JDWP is optional; it might not be available in some implementations of the Java™ 2 SDK. The existence of JDWP can allow the same debugger to work.

From Java 5.0 (or 1.5.0) onwards, specify the `-agentlib:jdwp` option to load and specify options to the JDWP agent. For Java releases prior to 5.0, the `-Xdebug` and `-Xrunjdwp` options are used. (The 5.0

implementation also supports the `-Xdebug` and `-Xrunjdwp` options but the newer `-agentlib:jdwp` option is preferable as the JDWP agent in 5.0 uses the JVMTI interface to the VM rather than the older JVMDI interface.) Specify this option in the correlator component in your launch configuration or when launching the correlator independently:

```
-J -agentlib:jdwp=transport=dt_socket,address=8787,server=y,suspend=n
```

This tells the JVM to use a socket transport on port `8787`. You can change this at will and use your own options. For more Java debugging/agent options see:

"http://docs.oracle.com/javase/1.5.0/docs/guide/jpda/conninv.html#Invocation" on page

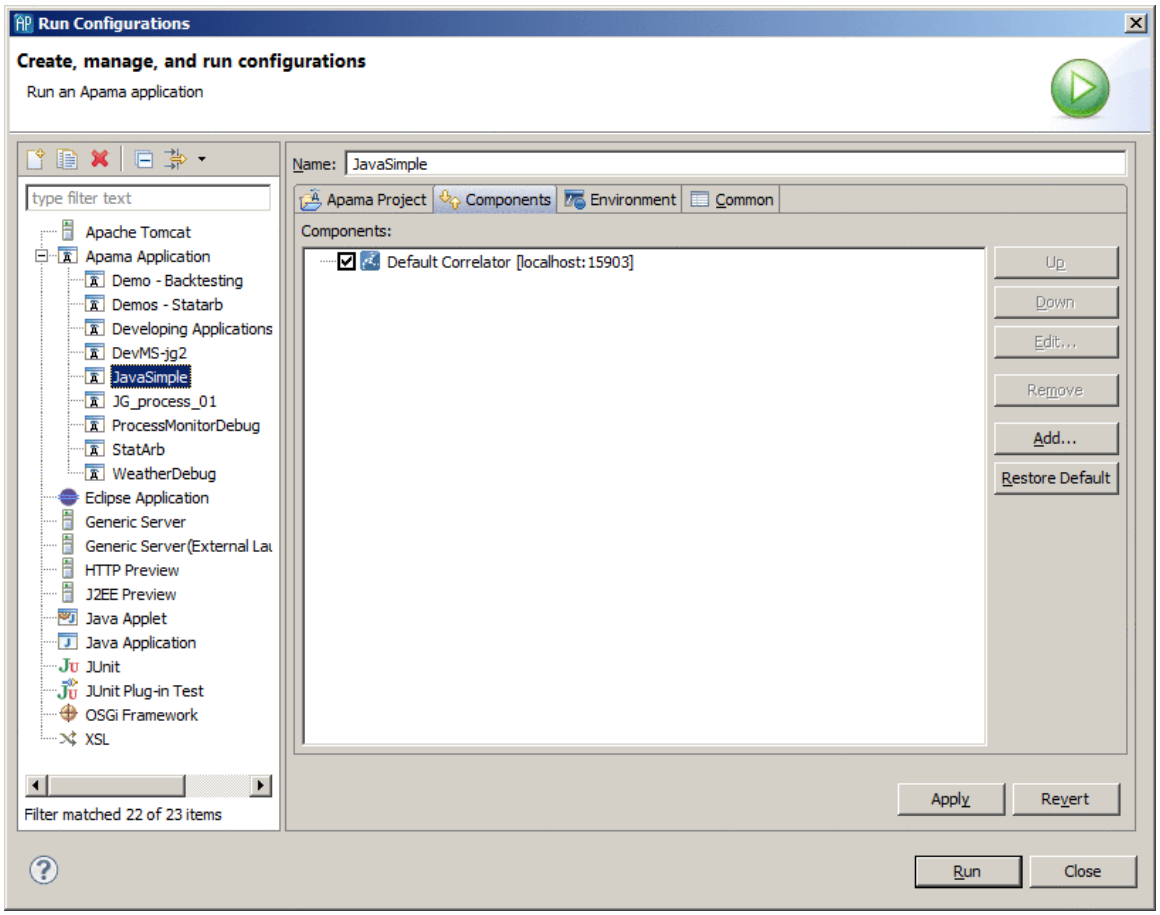
To launch the correlator with Java and remote debugging enabled, you can use a command line or Apama Studio.

On the command line, invoke the correlator with the following arguments:

```
correlator -l license.txt -j -J
-agentlib:jdwp=transport=dt_socket,address=8787,server=y,suspend=n
```

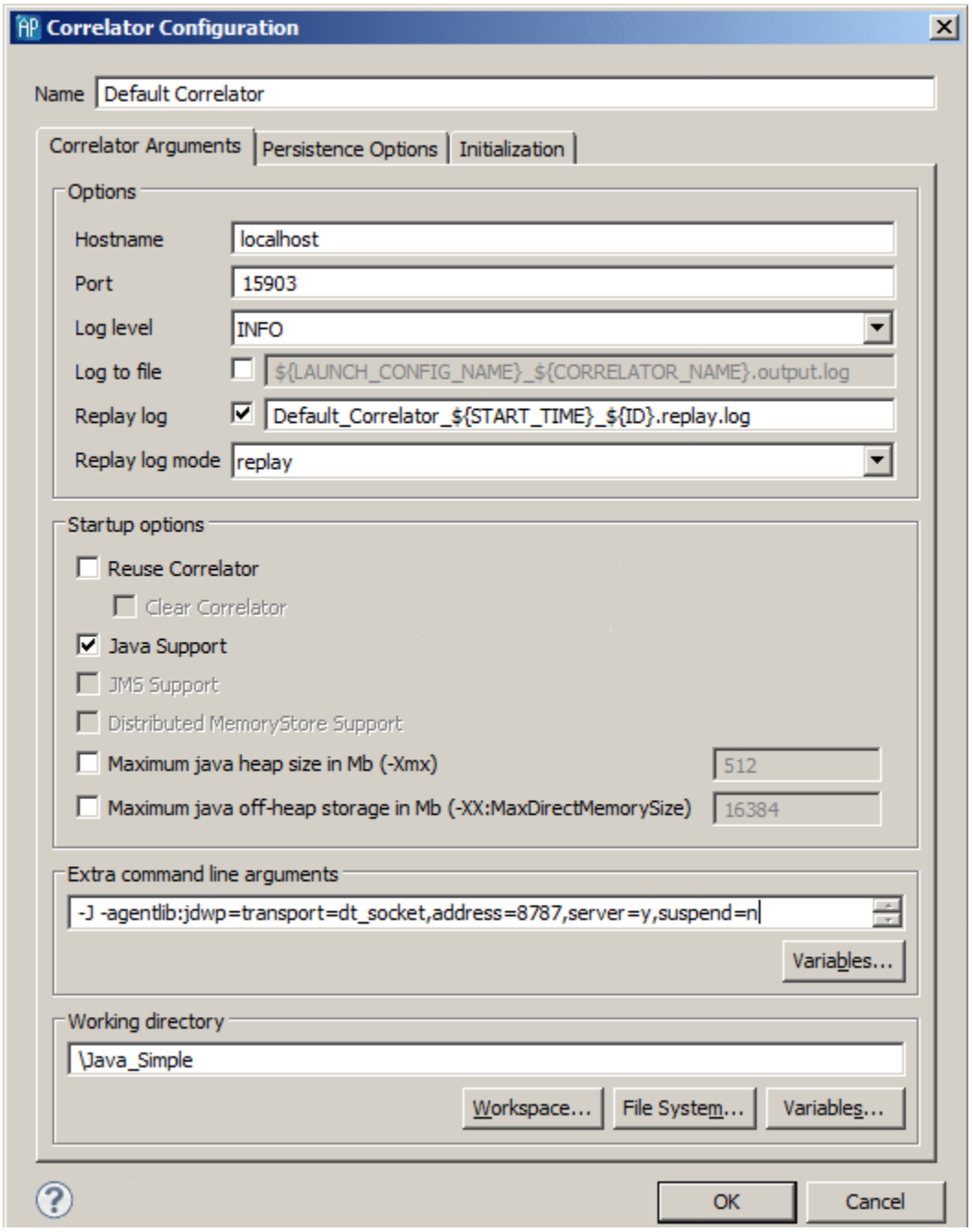
In Apama Studio, invoke the correlator as follows:

1. From the main Apama Studio menubar, select Run > Run Configurations.
2. Click the Components tab.
3. Select the correlator you want to start and click Edit.



4. Enter the following in the Extra command line arguments field:

   ```
   -J -agentlib:jdwp=transport=dt_socket,address=8787,server=y,suspend=n
   ```



5. Select OK and then Run to start the correlator.

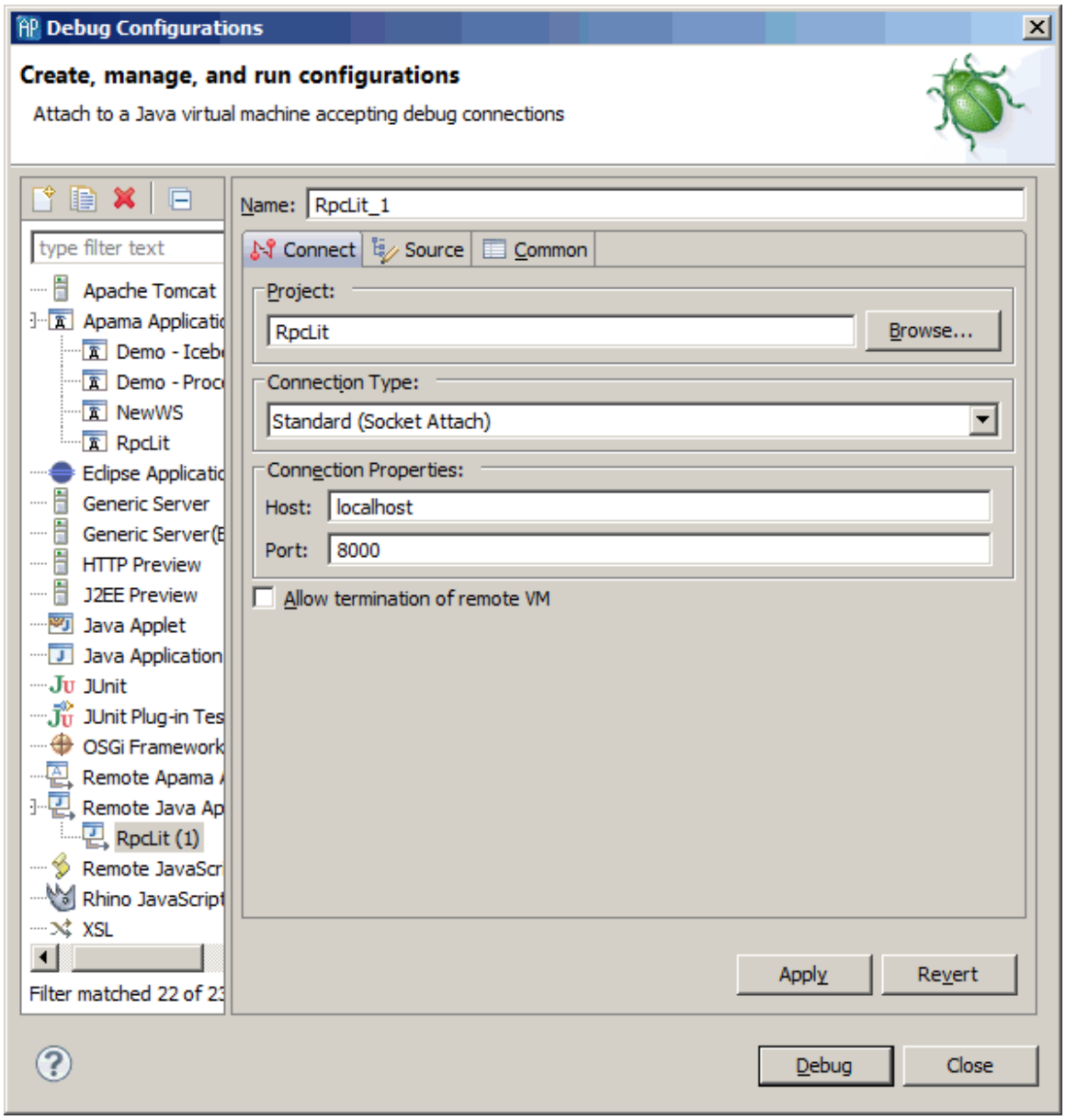
The correlator is now enabled for Java debugging. The next section discusses how to connect to the correlator with the Apama Studio debugger.

Debugging JMon Applications

Creating a debug run configuration

After you have a running correlator that is enabled for remote debugging, you must create a debug run configuration that you use to connect to the correlator:

1. From the Apama Studio menubar, select Run > Debug Configurations.
2. In the **Debug** dialog, on the left, select Remote Java Application.
3. From the **Debug** dialog toolbar, select New launch configuration .
4. Select the Connect tab, if it is not already selected.
5. Enter a name for your project and set the host and port for the remote JVM connection you previously specified as Java options when you started the correlator.



6. Click the Source tab and add the path to the Java source for your JMon application.
7. Click Apply and then Debug.

This launch configuration immediately connects to the remote JVM process in the correlator and Apama Studio switches to the **Apama Runtime**perspective, which you use for debugging. You are

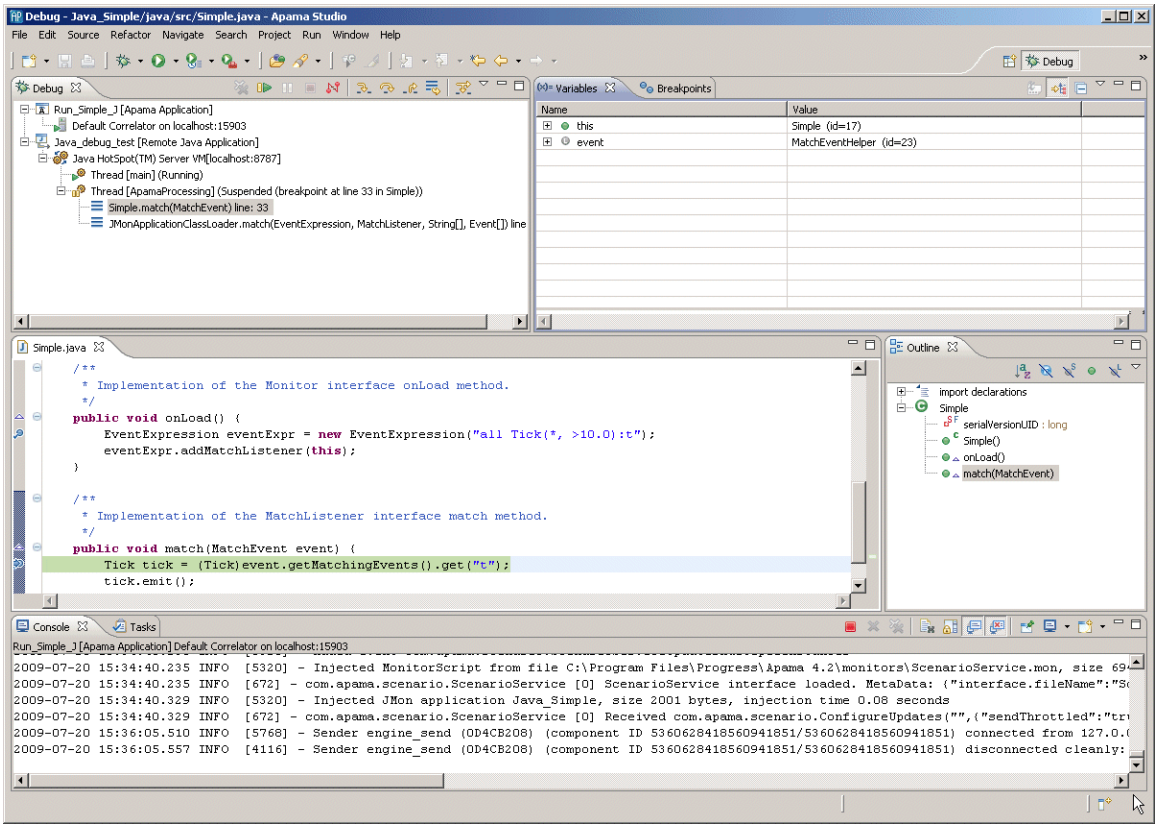
now ready to remotely debug your JMon application. Debugging a JMon application in Apama Studio is the same as debugging any other Java application in Eclipse.

Debugging JMon Applications

Debug perspective

The **Debug** perspective appears automatically when you start a debug session. The default **Debug** perspective has five panels that you can use for debugging:

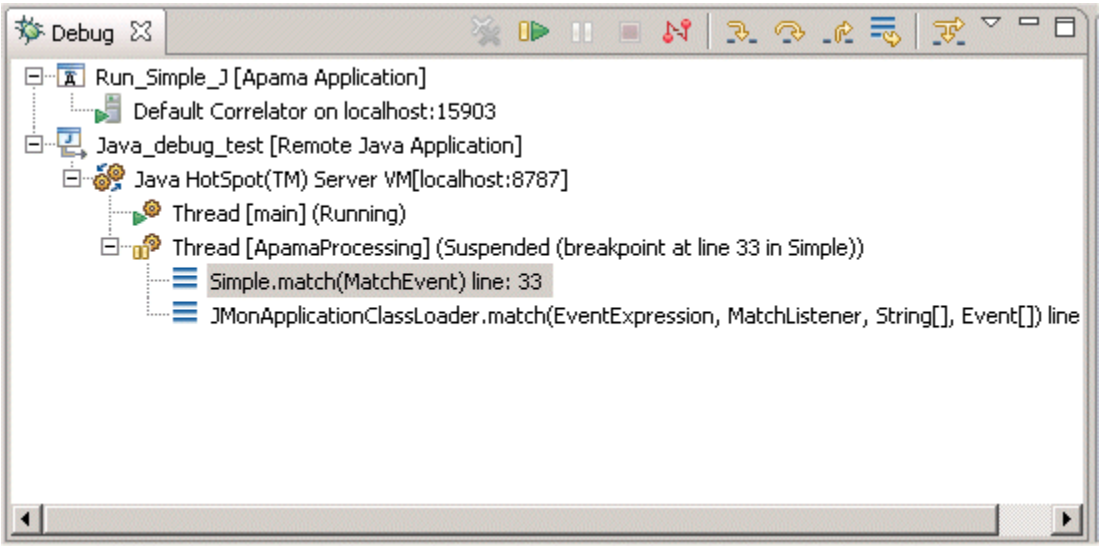
- The top-left panel is the **Debug** view which shows the application's stack frame. See "Using the Debug view" on page 139.
- The top-right panel provides a tabs for viewing the **Breakpoints** view and the **Variables** view. See "Working with breakpoints" on page 140 and "Viewing stack frame variables" on page 140.
- The middle-left panel is the code editor view.
- The middle-right panel is the code **Outline** view.
- The bottom panel includes the standard tabs for the console output, tasks, errors, problems, search results, display, and so on.



Debugging JMon Applications

Using the Debug view

The top left view is the main debug panel and shows the call stack and status of current threads, including any threads that have already run to completion. Whenever execution reaches a breakpoint, Apama Studio highlights the current thread.



This panel provides code execution controls. These allow you to suspend and resume execution, step through breakpoints in various ways, jump between stack frames, connect/disconnect to remote JVMs process, and so on. Following are descriptions of the most important controls:

	Resume	Select the Resume command to resume execution of the currently suspended debug target.
	Suspend	Select the Suspend command to halt execution of the currently selected thread in a debug target.
	Terminate	Ends the selected debug session and/or process. The impact of this action depends on the type of the item selected in the **Debug** view.
	Disconnect	Detaches the debugger from the selected process (useful for debugging attached processes).
	Step Into	Select to execute the current line, including any routines, and proceed to the next statement.
	Step Over	Select to execute the current line, following execution inside a routine.
	Step Return	Select to continue execution to the end of the current routine, then follow execution to the routine's caller.

	Drop to Frame	Select the Drop to Frame command to re-enter the selected stack frame in the **Debug** view.
	Use Step Filters	Select the Use Step Filters command to change whether step filters should be used in the **Debug** view.

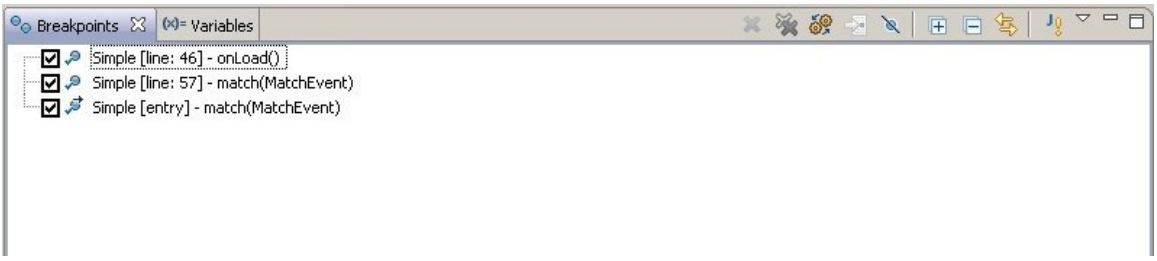
Debug perspective

Working with breakpoints

You can set breakpoints in your code in a number of ways. The most straightforward way is to select a line in your Java code and double click on the gray area to the left of the line. This sets a breakpoint before the execution of that line. The debugger will suspend execution when it reaches that line, allowing you to inspect variable values and results of expressions, either as defined in the code or created by you in real-time. These expressions can also be used in conditions for breakpoints.

You can set breakpoints on particular Java exceptions, on particular line numbers, on particular method calls, on conditions such as after a certain number of hits on a particular statement, and so on.

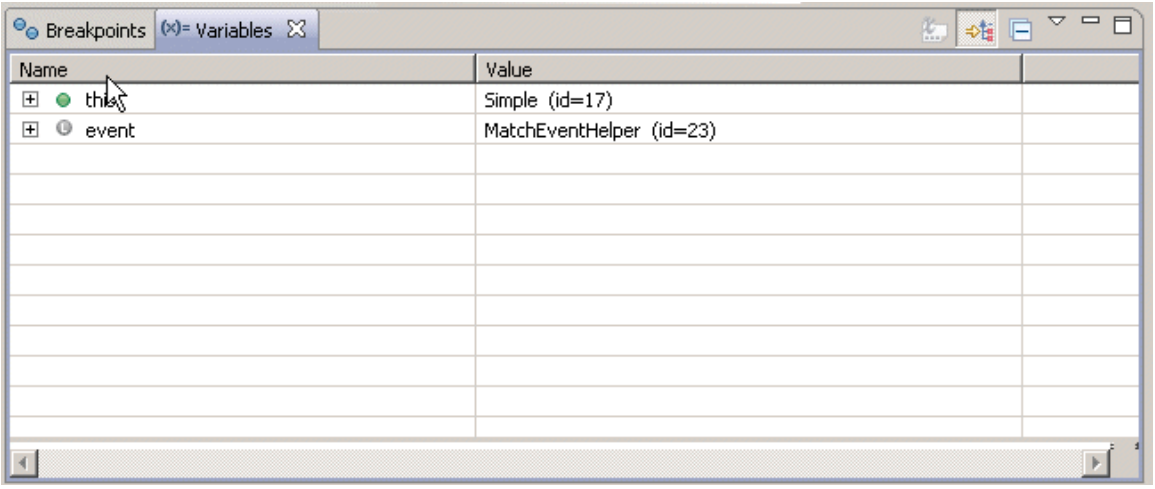
The breakpoint panel allows you to view and control all your currently defined breakpoints. After execution reaches a breakpoint, you can control process execution by stepping forwards or backwards through your code execution, or by simply resuming execution until the next breakpoint or until your program terminates.



Debug perspective

Viewing stack frame variables

When selecting a particular stack frame in the **Debug** view the variables panel on the top right side lets you inspect all variables as defined in the current stack context. Of course, you can also view variables alongside other variable values (globules, constants, and so on) if they are specified in the options. For more information, see the Eclipse debugging documentation.



Debug perspective

Example debug session

This section provides a very simple example that highlights some of the basic debugging concepts. It shows the basic steps for creating a JMon JAR file and injecting it into a debug enabled correlator. Familiarity with how to develop JMon applications is assumed and some steps are omitted based on this assumption.

The steps for developing a JMon application are as follows:

1. Implement the Java code.
2. Annotate the Java code with required JMon information.
3. Generate the classes from the code.
4. Package the classes into a JAR file.
5. Create the JMon JAR manifest manually or by using the `JarProcessor` utility.
6. Inject the JAR file into a running correlator that is enabled with the JVM and with the proper Java options for remote debugging.

This section provides details as follows:

- "Debug example of preparing code and JAR file" on page 141
- "Debug example of starting correlator and injecting application" on page 143
- "Example of debugging in Apama Studio" on page 143

Debugging JMon Applications

Debug example of preparing code and JAR file

Consider the following sample Java code, which is the basis for a JMon application. The files containing the complete code are `Simple.java` and `Tick.java`, which are located in the `samples\java_monitor\simple\src` directory of your Apama installation.

```
@Application(name = "Simple",
          author = "John Doe",
          version = "1.0",
          company = "Apama",
          description = "sample")
@MonitorType(description = "A simple Java monitor")

public class Simple implements Monitor, MatchListener {
   public Simple() {}
   public void onLoad() {
      EventExpression eventExpr =
         new EventExpression("all Tick(*, >10.0):t");
      eventExpr.addMatchListener(this);
   }

   public void match(MatchEvent event) {
      Tick tick = (Tick)event.getMatchingEvents().get("t");
      tick.emit();
   }
}
```

Here is the `Tick` class (`Tick.java`):

```
@EventType(description = "Event that signals a stock trade")
public class Tick extends Event {
   public String name;
   public double price;
   public Tick() {
      this("", 0);
   }

   public Tick(String name, double price){
      this.name = name;
      this.price = price;
   }
}
```

Assume that `Simple.java` is part of your Apama Studio project. Open or create the project in Apama Studio and add the code if necessary. Note that this code is already annotated with the information required for creating the JMon `JAR` manifest.

The next step is to compile and pack the class bytecode into a `JAR` file, and create the manifest that flags and describes this `JAR` file as a JMon application. You can do this by setting up a new builder in the project properties in Apama Studio or through the command line. For the sake of simplicity, the standard command line calls are described below:

1. Compile the Java code. For example:

   ```
   javac -classpath "%APAMA_HOME%/lib/correlator_extension_api5.2.jar;%APAMA_HOME%/lib/util5.2.jar"
   *.java
   ```

2. Package the classes into your `JAR` file:

   ```
   jar cf simple-jmon.jar *.class
   ```

3. Use the `JarProcessor` utility to create the manifest for the `JAR` file. For example, if the `JAR` is called `simple-jmon.jar`, enter:

   ```
   java -classpath "%APAMA_HOME%/lib/correlator_extension_api5.2.jar"
   com.apama.jmon.annotation.JarProcessor simple-jmon.jar
   ```

Debug example of starting correlator and injecting application

Start a local correlator with remote debugging enabled. For example, if you specify port `8787`, the command line looks like this:

```
correlator -l license.txt -j -J
-agentlib:jdwp=transport=dt_socket,address=8787,server=y,suspend=n
```

Inject the `JAR` into the running correlator. For example:

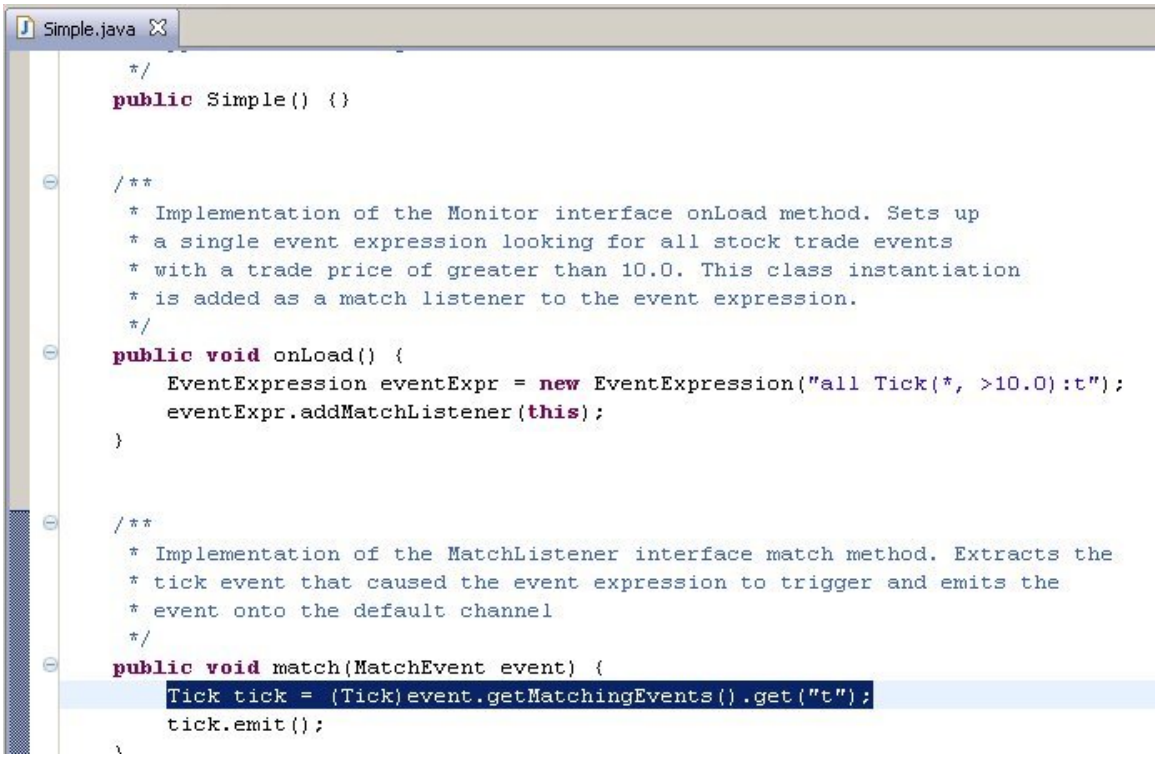
```
engine_inject -j simple-jmon.jar
```

You are ready to start an Apama Studio debug session.

Example of debugging in Apama Studio

You want to connect to the remote correlator and debug the JMon application in Apama Studio. To do this, you must open the JMon project (or the Apama project with Java support enabled) in Apama Studio and create a debug launch configuration. This launch configuration connects to the remote Java application, which is the JVM in the running correlator.

For test purposes, suppose that you create a simple breakpoint in the `match()` method in `Simple.java`. This suspends execution when the correlator reaches a matching event listener.

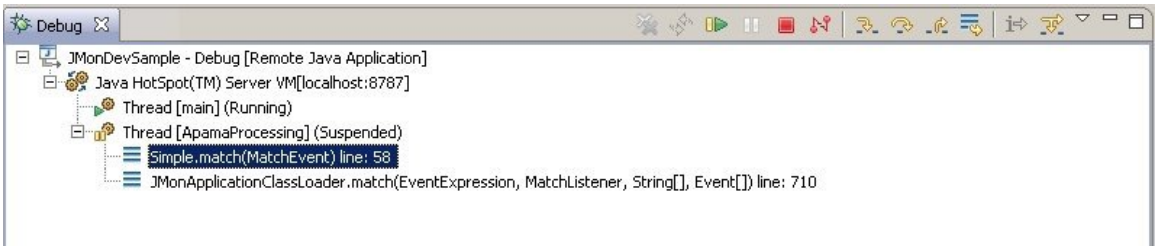


Send the following events to the correlator to trigger the breakpoint:

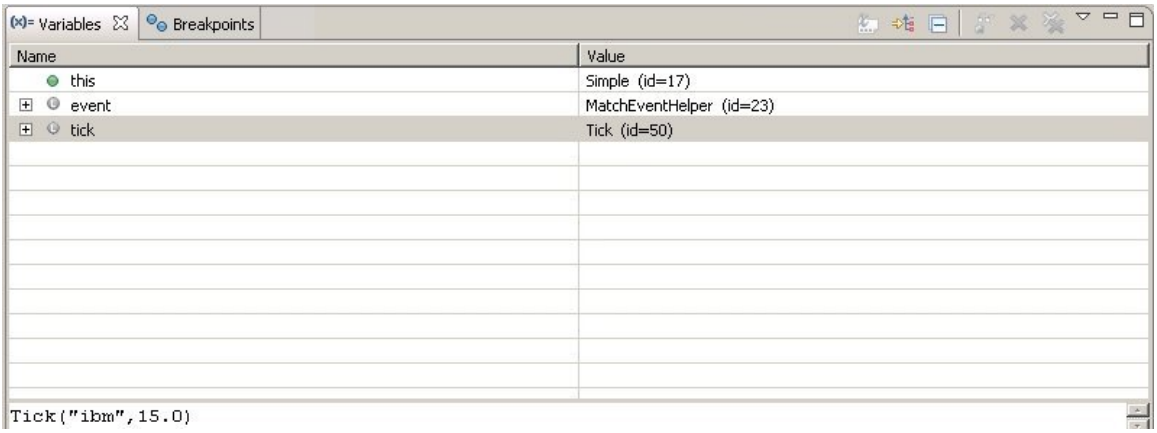
```
Tick("ibm", 1.0)
Tick("ibm", 5.0)
Tick("ibm", 15.0)
Tick("msft", 15.0)
```

As you can see, the debugger stops execution at the specified code breakpoint after the listener finds the first `Tick` event with a price greater than `10.0`, that is `Tick("ibm", 15.0)`.

Suppose that you want to examine the heap context, that is, the values of the variables. You can observe this in the top left Debug panel of the **Apama Runtime** perspective. Select the Simple.match method stack frame from Thread[ApamaProcessing]. For example:



Note that the top right **Variables** view now shows the proper stack frame context with all relevant heap space variable values. The `tick` variable, defined in the code, is not yet visible. This is because execution was suspended before the current line was executed. To execute the current line, which extracts the matching `Tick` event and assigns it to the `tick` variable, click Step Over in the Debug panel toolbar. As you can see, the tick variable now appears in the **Variables** view. You can select it to inspect its value, which is, of course, `Tick("ibm", 15.0)`.



Additional resources for Java debugging

This section introduces the basic concepts for debugging JMon applications in Apama Studio. There are a number of Apama Studio debugging features not mentioned, but the mechanisms are essentially the same as for other Java applications. For additional information about available debugging options, see the following Eclipse information:

- Running and debugging Java in Eclipse:

  http://help.eclipse.org/kepler/index.jsp?topic=/org.eclipse.jdt.doc.user/tasks/task-running_and_debugging.htm

- Eclipse And Java: Using the Debugger (Video Tutorial):

  http://www.eclipse.org/resources/resource.php?id=405

- JDB (Java debugger):

http://docs.oracle.com/javase/1.5.0/docs/tooldocs/windows/jdb.html

- Good video tutorials and practice exercises for debugging Java in Eclipse: http://eclipsetutorial.sourceforge.net/debugger.html
- IBM debugging with Eclipse: http://www.ibm.com/developerworks/library/os-ecbug/

Debugging JMon Applications

Chapter 7: Profiling EPL Applications

You can profile Apama applications written in EPL with Apama Studio. Data collected in the profiler allows you to identify possible bottlenecks in an EPL application. The Apama Studio profiler consists of the **Apama Profiler** perspective which includes a variety of views. With the Apama Studio EPL Profiler, you can profile applications running on both local and remote correlators.

Launching profiling sessions

You can profile an Apama EPL application using a default launch configuration. You can also create custom profile launch configurations for launching profiling session. For example, you may want the profiler to launch the application using a correlator different from the project's default correlator or to profile an application running on a remote correlator.

- When a default profile configuraton is launched, it starts the profiler and the default correlator and the application if they aren't already running. If they are already running the profiler just starts profiling the running application. See "Launching a default profiling session" on page 147 for more information on launching a default profiling session.
- If you want to profile an application that runs on a non-default correlator, you need to create a launch configuration that points to the run configuration associated with the desired correlator. See "Launching a custom profiling session" on page 147 for more information for creating and launching a custom profiling configuration.
- If you want to profile an application that runs on a remote correlator, you need to create a launch configuration that points to the machine where the correlator is running. In order to profile an application running on a remote correlator, the remote application needs to be running before you launch the profiling session. See "Launching a remote profiling session" on page 148 for more information on creating and launching a configuration for a remote profiling session.

Profiling EPL Applications

Launching a default profiling session

You can profile an Apama EPL application using a default launch configuration or you can create a custom profiler launch configuration for an application running on a remote correlator.

To launch a profiling session in Apama Studio using the default launch configuration:

1. From the Apama Studio menu, select Run > Profile.
2. By default, Apama Studio displays the **Confirm Perspective Switch** dialog asking if you want to use the **Apama Profiler** perspective. Click Yes. Apama Studio switches to the **Apama Profiler** perspective and begins collecting data from the EPL application.

   Note, you can change the behavior of the **Confirm Perspective Switch** dialog by changing a setting in the **Profiling Monitor** view's Preferences wizard; see "Displaying Apama perspective preferences" on page 157.

Launching a custom profiling session

You can create custom launch configurations for profiling Apama EPL applications. See "Creating a custom profile launch configuration" on page 147 for more information on creating a custom profiler launch configuration.

To launch a profiling session in Apama Studio using a custom launch configuration, assuming that you have defined a custom profiler launch configuration:

1. In the **Project Explorer** right-click the project name and select Profile As > Apama Correlator Profiler. The **Select Profile Application** dialog is displayed.
2. In the **Select Profile Application** dialog, select the launch configuration you want to use and click OK.
3. By default, Apama Studio displays the **Confirm Perspective Switch** dialog asking if you want to use the **Apama Profiler** perspective. Click Yes. Apama Studio switches to the **Apama Profiler** perspective and begins collecting data from the EPL application.

   Note, you can change the behavior of the **Confirm Perspective Switch** dialog by changing a setting in the **Profiling Monitor** view's Preferences wizard; see "Displaying Apama perspective preferences" on page 157.

Launching profiling sessions

Creating a custom profile launch configuration

You can create custom profile launch configurations for launching profiling session. For example, you may want the profiler to launch the application using a correlator different from the project's default correlator.

1. In the **Project Explorer** right-click the project and select Profile As > Profile Configurations. The **Profile Configurations** wizard starts.
2. In the **Profile Configurations** wizard, click the New launch configuration button.

3. In the **Profile Configurations** wizard, replace the default name in the Name field if desired.
4. In the **Profile Configurations** wizard, on the Connection Details tab, in the Profile a field, use the default Apama Launch configuration selection.
5. In the Launch Configuration field, click the Browse button. The **Choose Launch Configuration** dialog is displayed.
6. In the **Choose Launch Configuration** dialog, select the run configuration you want to profile and click OK.
7. In the **Profile Configurations** wizard, click Apply to save the profile configuration or Profile to save and launch the profiling session.

Launching a custom profiling session

Launching a remote profiling session

In Apama Studio you can profile an Apama application running on a remote correlator by creating a custom profiler launch configuration that points to the remote machine where the correlator is running.

Before you launch a profiling session for an application running on a remote correlator, the remote application needs to be already running. To launch a profiling session for an application on a remote correlator, assuming that you have already created a remote profiler launch configuration:

1. In the **Project Explorer** right-click the project name and select Profile As > Apama Correlator Profiler. The **Select Profile Application** dialog is displayed.
2. In the **Select Profile Application** dialog, select the launch configuration you want to use to connect to the remote correlator and click OK.
3. By default, Apama Studio displays the **Confirm Perspective Switch** dialog asking if you want to use the **Apama Profiler** perspective. Click Yes. Apama Studio switches to the **Apama Profiler** perspective and begins collecting data from the EPL application.

   Note, you can change the behavior of the **Confirm Perspective Switch** dialog by changing a setting in the **Profiling Monitor** view's Preferences wizard; see "Displaying Apama perspective preferences" on page 157.

Launching profiling sessions

Creating a remote profiler launch configuration

You can create custom profile launch configurations for profiling Apama applications running on remote correlators.

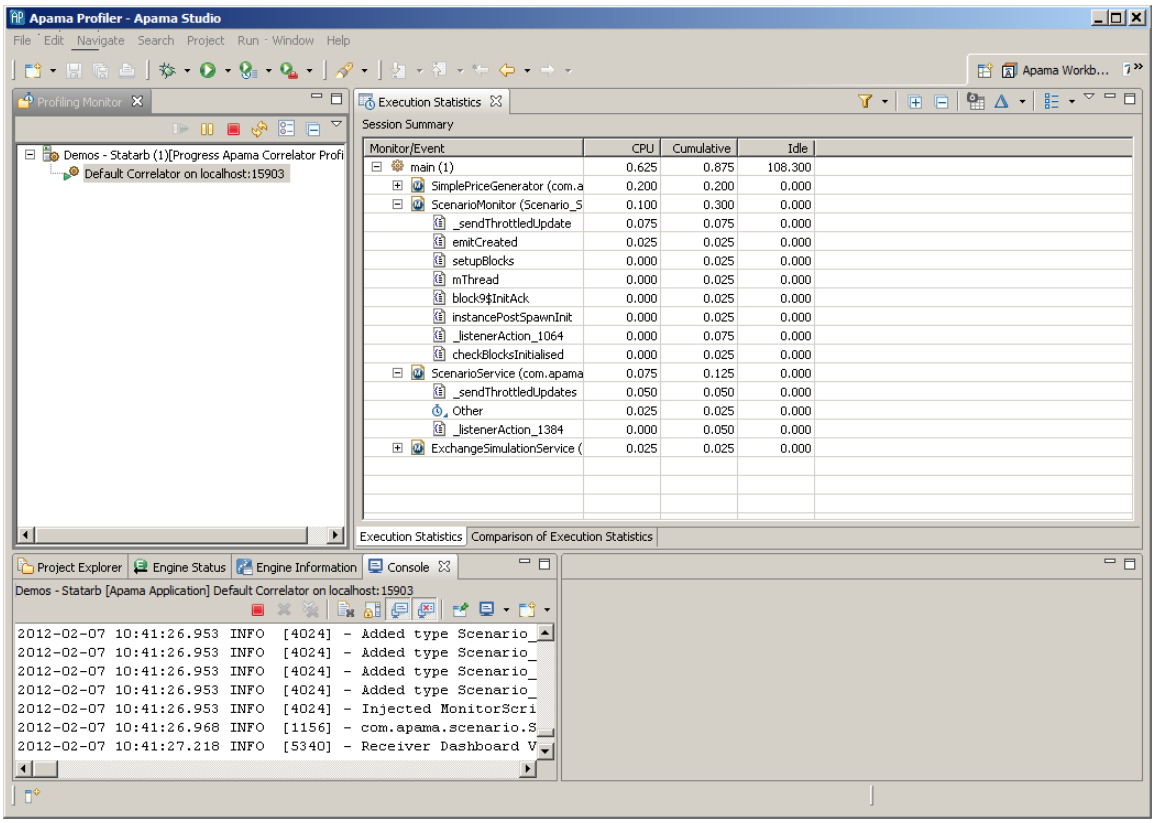
1. From the **Project Explorer**, right-click the project and select Profile As > Profile Configurations. The **Profile Configurations** wizard is displayed.
2. In the **Profile Configurations** wizard, enter a meaningful name for the configuration in the Name field if you do not want to use the default name.
3. On the Connection Details tab, in the Profile a field, select Remote Apama correlator.
4. In the Remote Correlator Details section, enter the name of the host and the port in the Host and Port fields.

5. If desired, and if the remote correlator is running, click Test Connection to confirm the specified location information is correct.
6. Click Apply to save the profile configuration or Profile to save and launch the profiling session.

Launching a remote profiling session

The Apama Profiler perspective

The following illustration shows an active profiling session.



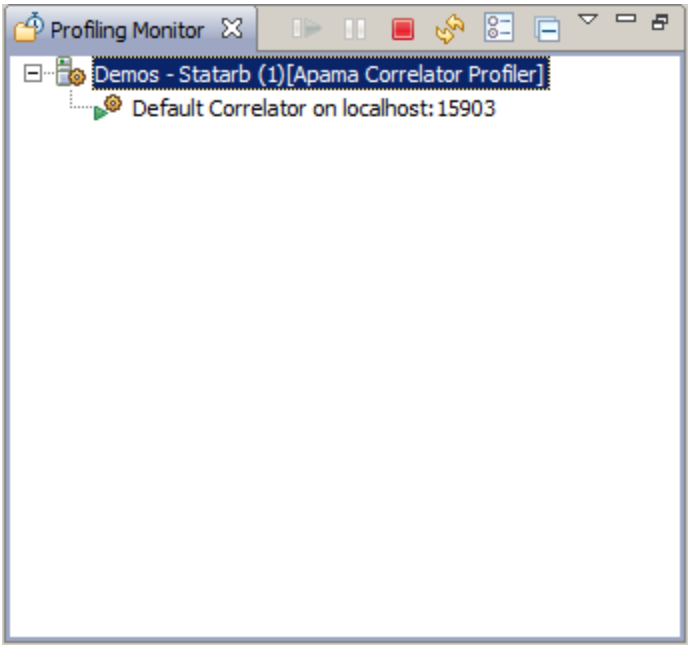
The **Apama Profiler** perspective consists of the following views:

- **Profiling Monitor** view
- **Execution Statistics** view

Profiling EPL Applications

Profiling Monitor view

The **Profiling Monitor** view shows a tree view of the available profiler sessions. These sessions could be associated with different applications or associated with applications running on different correlators.



The tool bar for the **Profiling Monitor** view contains the following buttons:

- — Resume the profiling session.
- — Pause the profiling session.
- — Stop the profiling session.
- — Manually refresh the **Execution Statistics** view with collected data. By default the data is automatically refreshed at 10 second intervals; you can change the refresh behavior with the Preferences button described below.
- — Displays a **Preferences** wizard where you can change the following:
  - Specify whether a profiler automatically displays the **Apama Profiler** perspective.
  - Specify how often to refresh the **Execution Statistics** view with new data.

  For more information on refreshing profiler data, see "Updating profile data" on page 156.
- — Collapse the entries in the tree view displayed in the **Profiling Monitor** view.
- — Display the **Profiling Monitor** view's drop down menu.

The Apama Profiler perspective

Execution Statistics view

The **Execution Statistics** view displays the timing details of the profiled EPL application. This view includes the following tabs:

- Execution Statistics
- Comparison of Execution Statistics

You can adjust the way the information is displayed using the buttons on the view's tool bar. The following tools are available:

-  — Apply and manage filters for displaying subsets of the profiling data. This tool is not available for the Comparison of Execution Statistics tab. For more information on filtering the data, see "Using filters" on page 153
-  — Expand the entire of tree of entries.
-  — Collapse the entire of tree of entries.
-  — Take a snapshot of the profiling data.
-  — Manage snapshots.
-  — Specify how to organize the profiling information by Contexts, Monitors/Events, or Action.
- — Choose which columns to display. You can also change the order in which the columns are displayed.

The Apama Profiler perspective

The Execution Statistics tab

The Execution Statistics tab carries a sub-title of "Session Summary". By default the left-hand column displays a tree view of the application's contexts, monitors, and actions organized by contexts. You can change the display to show profiler information organized by monitors/events and by actions using the Organize View By button [].

You can filter the profiler information in order to focus on specific application behavior using the Filter button []. For more information on using filters, see "Using filters" on page 153.

By default the display includes the following three columns of information:

- CPU time — The time in seconds this action has been executing.
- Cumulative time — The time in seconds spent in this action and all its child actions.
- Idle — The time in seconds this context has been idle, waiting for events.

You can also display the following information, by using the Choose Columns button [[].

- Empty — The time in seconds this context has been empty, that is, without monitors.
- Non-Idle — The time in seconds this context has been non-idle, in other words, when it has had events to process.

- Runnable — The time in seconds this context has had work to do, but during which the work has not been executed by the scheduler.
- Plugin — The time in seconds spent in a plugin call.
- Blocked — The time in seconds this context has been blocked, for example, waiting for a queue to become non-full.

Execution Statistics view

Comparison of Execution Statistics tab

This tab compares the most recent profiling data with that of a previous set. For each type of information there are three associated columns, for example CPU, Old CPU, and Diff CPU. The leftmost column is the current information, the column labelled "Old" is the information from the previous set, and the column labelled "Diff" is the change between the new and the previous information. The previous set is considered the baseline.

By default, the "previous set" consists of the data from the last time it was refreshed. You can change this behavior to compare the most recent data to data captured in a snapshot at a particular moment in time. For information on capturing data in a snapshot, see "Taking snapshots" on page 155

As with the Execution Statistics tab, you can change the type of information and how it is displayed on the Comparison Execution Statistics tab as follows.

- Use the Organize View By button [] to organize the profiler information by Contexts, Monitors/Events, or Actions.
- Use the Choose Columns button [] to add, remove, or change the order of the information columns.

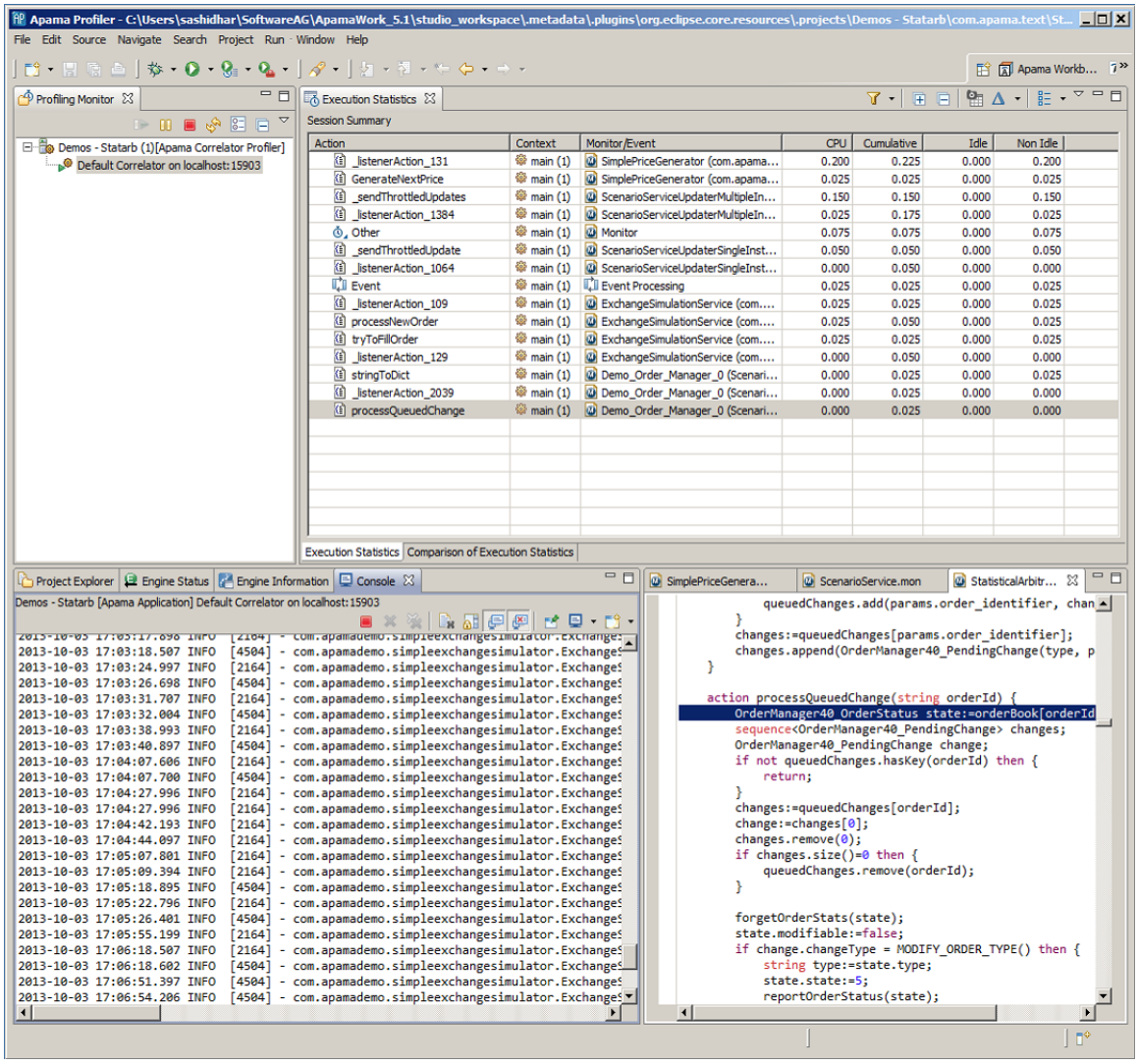
Execution Statistics view

Viewing EPL code

In the **Execution Statistics** view you can display the EPL code for a specific action or listener. Only the action-level entries that are shown with the code icon [] are able to display the associated code. There are other entries, such as garbage collector timings (that is, GC-mark, GC etc.), events and others for which there is no code association.

To display the code for an action or listener:

In the **Execution Statistics** view, double-click the action or listener for which you want to view the code. The EPL source file containing the code for the action or listener opens (if it is not already open) in the EPL editor.



The Apama Profiler perspective

Using filters

When viewing the Execution Statistics tab of the **Execution Statistics** view, you can focus on specific data by applying a filter to the profiling information.

Profiling EPL Applications

Creating a Filter

You can create filters in order concentrate on particular profiling data. To create a filter:

1. On the Execution Statistics tab, click the Filter button [] on the view's tool bar and select Manage Filters. The **Manage Filters** dialog is displayed.
2. In the **Manage Filters** dialog, click Add. The **Create Filter** dialog is displayed.

3. In the **Create Filter** dialog, specify the Filter Name and an optional Filter Description.
4. In the **Create Filter** dialog, click the Add New Row button []. This adds a row in the table field as indicated by a new set of parenthesis marks ( ).
5. Specify the filter criteria as follows:
   a. Click the Type column and from the drop-down list select the type of data you want to filter for.
   b. Click on the Operation column and from the drop-down list specify the operation that is appropriate to the Type you are filtering for.
   c. Double-click in the Value column and enter an appropriate value; then click in a blank area of the table to accept the value you specified.
   d. You can add other criteria to the filter by ANDing or ORing. In the AND/OR column, select the appropriate operator from the drop-down list, click the Add New Row and specify the additional filter criteria as described above.
6. Click OK.
7. In the **Manage Filters** dialog, select the filter you want and click Apply.

   The Execution Statistics tab is updated according to your filter criteria.

Using filters

Managing Filters

Use the Filter tool to manage the use of filters. You can add, edit, or remove filters; change which filter is used; and revert to the default display of unfiltered data. To manage filters:

1. On the Execution Statistics tab, click the Filter tool [] and select Manage Filters from the pop-up menu. The **Manage Filters** dialog is displayed.
   - If you want to create a new filter click Add. The **Create Filter** dialog is displayed. See "Creating a Filter" on page 153 for more information on creating filters.
   - If you want to edit an existing filter click, select the name of the filter and click Edit. The **Edit Filter** dialog displays the current filter criteria. Edit the filter information as necessary and click OK.
   - If you want to remove a filter, select the name of the filter and click Delete.
2. Select the filter you want and click Apply. If you remove all the filters and click Apply, the profiler displays shows the default unfiltered set of information.

If you are looking at a filtered data set and want to return to the default display of unfiltered data, click the Filter tool [] and select Reset Filter from the pop-up menu.

Using filters

Taking snapshots

You can capture profiling data at any moment in time by taking snapshots. You can then compare current profiling data to the data represented in a snapshot. To take a snapshot:

1. In the **Execution Statistics** view, click the Take Snapshot button []. The **Create Snapshot** dialog is displayed.
2. Enter a meaningful name in the Snapshot Name field.
3. Specify the source of the data you want to use to create the snapshot as follows:
   - To populate the snapshot with data that the correlator captures at this point in time, select Current correlator.
   - To populate the snapshot with data that has previously been stored on disk in a `.csv` file select Select from file. This data would typically be collected using the `engine_management` tool as, for example

     ```
     engine_management -r "profiling get" > myfile.csv
     ```

     Note, however to be used this way, the `.csv` file must have been generated using Apama release 5.0.1 or later.

     For more information on the `engine_management` utility, see "Shutting down and managing components" in *Deploying and Managing Apama Applications*.
4. Click OK.

In the Comparison of Execution Statistics tab the baseline columns (labelled "Old") display the data captured in the snapshot and the columns labelled "Diff" are similarly updated to compare the current information with that captured in the snapshot. For more information on managing snapshots, see "Using snapshots" on page 155.

Profiling EPL Applications

Using snapshots

After you create a snapshot the display on the Comparison of Execution Statistics tab is updated to compare the current data with that captured in the snapshot. You can change this to compare the current data to another snapshot or the default setting that compares the current data to the data seen the previous time the data was refreshed. To change the compared sets of data:

In the Comparison of Execution Statistics tab, click the Show Previous Snapshots button [].

- If you want to use the data from another snapshot, select the name you assigned to it.
- If you want to return to the profiler's default comparison, select Compare last data.
- If you want to manage your snapshots
  1. Select Manage snapshots.
  2. Select the name of the snapshot you want to apply or remove.

3. Click OKto apply the snapshot or Remove to remove the snapshot.

- If you want remove all snapshots, select Clear Snapshots.

Profiling EPL Applications

Choosing profiling information columns

You can specify which information to display in the **Execution Statistics** view. You can also change the order in which the columns are displayed. By default, the following columns are displayed:

- CPU — The time in seconds this action has been executing.
- Cumulative — The time in seconds spent in this action and all its child actions.
- Idle — The time in seconds this context has been idle, waiting for events.

1. In the **Execution Statistics** view, click the View menu button [ ] and select Choose columns. The **Choose Columns** dialog is displayed.
2. Add a check to the check boxes that correspond to the columns you want to display.
3. Click OK.

The **Execution Statistics** view is updated to show all the selected the columns.

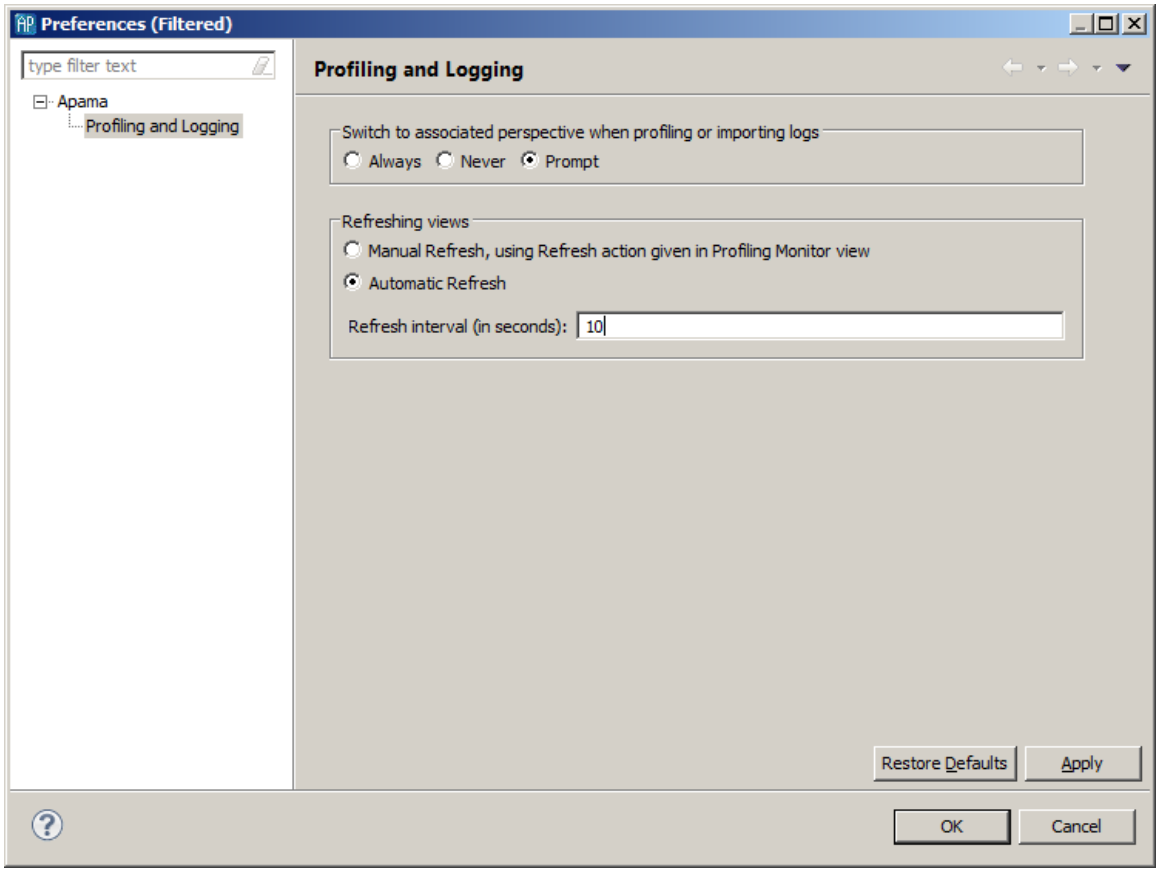
Profiling EPL Applications

Updating profile data

By default, the Apama Studio profiler polls for data every 10 seconds and then updates the display in the **Execution Statistics** view. You can change the polling frequency. You can also change the way the profiler updates the data from a polling mode (Automatic Refresh) to an on-demand mode (Manual Refresh). Using the on-demand mode is useful, for example, if you want to let your application run up to a certain point and then retrieve profile data or if you want to inject a set of test events and then examine the profile deltas.

To change the way the profiler updates data:

1. In the **Profiling Montior** view, click the Preferences button [ ]. The **Preferences** wizard is displayed.



2. In the **Preferences** wizard, select the Apama > Profiling and Logging node in the left pane. In the right pane, in the Refreshing views field, select the behavior you want and click OK.

   If you select Automatic, you can also specify the refresh interval.

Profiling EPL Applications

Displaying Apama perspective preferences

By default, when you launch a profiling session, Apama Studio displays the **Confirm Perspective Switch** dialog asking if you want to switch to the **Apama Profiler** perspective.

You can change the this behavior by changing the setting in the **Profiling Monitor** view's Preferences.

To change this behavior:

1. In the **Profiling Monitor** view, click the Preferences icon [ ]. This displays the **Preferences** wizard.
2. In the **Preferences** wizard, select the Apama > Profiling and Logging node in the left pane. In the right pane, in the Switch to associated perspective when profiling or importing logs field, select the behavior you want and click OK.

Profiling EPL Applications

Chapter 8: Using the Data Player

With the Apama Data Player you can play back previously saved event data as you develop your application. During playback, you can analyze the behavior of your application. Or, if you modify the saved event data, you can analyze how your application performs with the altered data. Apama Studio plays back event data that has been stored in standard data formats.

Introduction to the Data Player

The Data Player relies on Apama Database Connector (ADBC) adapters that are specific to standard ODBC and JDBC database formats as well as the proprietary Apama Sim format. These adapters run in the Apama Integration Application Framework (IAF), which connects the data sources to the correlator.

- For more information on ADBC, see "Using the Apama Database Connector" in *Deploying and Managing Apama Applications*.
- For more information on the IAF, see "The Integration Adapter Framework" in *Developing Adapters* (available if you selected Developer during installation.

The Apama Data Player consists of both the Query Editor and the Data Player Control.

Using the Data Player

Using the Data Player

In addition to the normal operations for running an application in Apama Studio, in order to play back event data in the Data Player, you need to perform a few other steps. Broadly, these steps are:

1. Configure the Apama Studio project to use the appropriate ADBC adapter for the data source and database and specify the event types that will be played back along with the appropriate IAF mapping.
2. Launch the project so it can establish a connection to the data source.
3. Specify a playback query to determine what data from the database you want to play back.
4. Use the Data Player control to specify the following: how fast you want to play back the data; over what time range; and what throttling period to use.
5. Run the playback session.

Adding the ADBC adapter

If you want to use the Apama Data Player in your project, you need to add and configure the Apama Database Connector adapter that is appropriate to the data source used by the project: ODBC, JDBC, or Sim.

1. There are two ways of adding an ADBC adapter to a project.

   If you are creating a new Apama project, select File > Project > New > Apama , give it a name, and click Next.

   If you are adding an ADBC adapter to an existing project:

   a. In the Project Explorer right-click the project and select Apama > Add Adapter. The **Add Adapter Instance** dialog opens.

   b. If desired, in the **Add Adapter Instance** dialog, create a new name for the adapter instance or accept the default instance name. Apama Studio prevents you from using a name that is already in use.

2. In the **New Apama Project** dialog or the **Add Adapter Instance** dialog, select the ADBC adapter bundle that is appropriate to the kind of data source your application will use. Click Finish or OK.

Apama Studio adds an `Adapters` node to your project if not already present. The node will contain a node for the new adapter. The adapter node contains an instance of the new adapter.

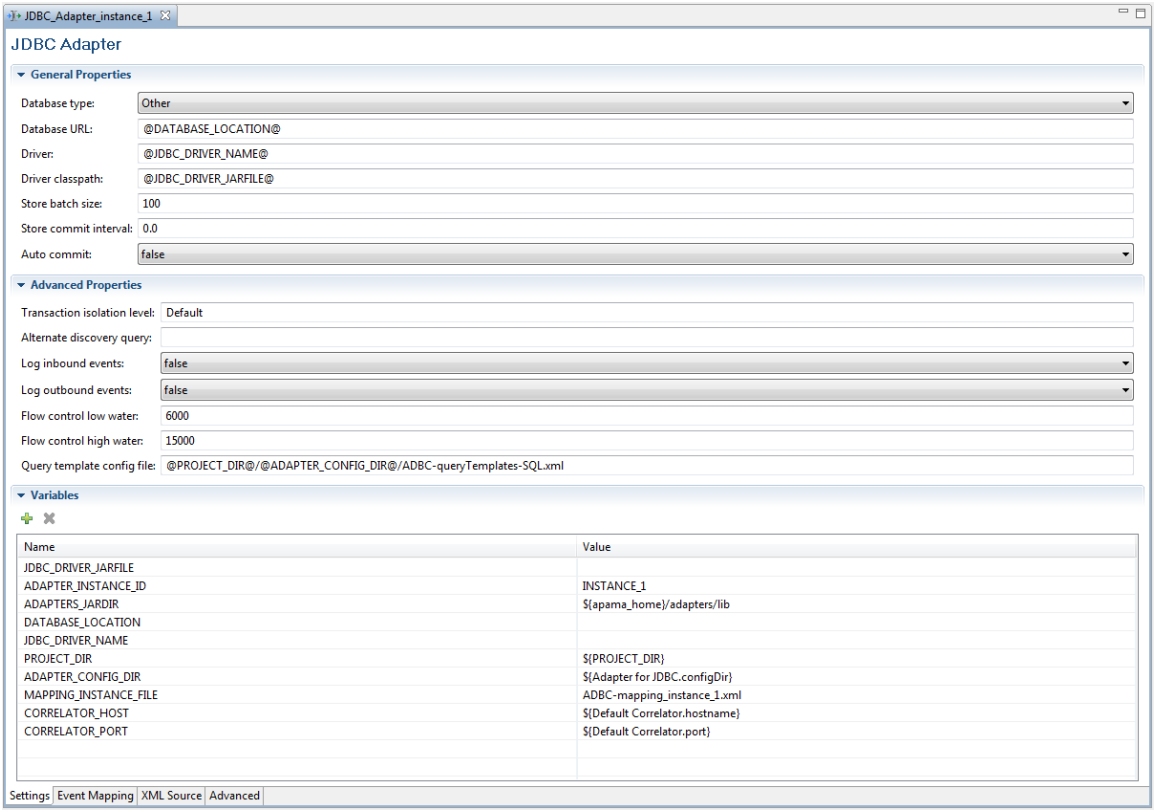
When you add a data source-specific adapter, the ADBC Adapter common (Apama Database Connector Common common) bundle will be added to the project automatically.

Configuring the ADBC adapter

To configure an instance of an ADBC adapter:

1. In the Project Explorer, expand the project's `Adapters` node and open the adapter folder (either `Adapter for ODBC`, `Adapter for JDBC`, or `Adapter for Sim`).

2. Double-click the entry for the adapter instance you want to configure. The configuration file opens in the adapter editor. For example, a configuration file for an instance of the ADBC-JDBC adapter looks like this:



The editor's graphical display presents configuration information in three separate sections:

- General Properties
- Advanced Properties
- Variables

For an instance of the ADBC-ODBC adapter, the display is similar but with fewer items in the General Properties and Variables section. For an instance of the ADBC-Sim adapter, the display only shows the Variables section.

For more information on specifying ADBC properties and variables, see "Configuring an ADBC adapter" in *Deploying and Managing Apama*.

Launching the project

To create queries most efficiently, the project needs to be running so that you can see what data sources, databases, and existing queries are available. If you need to create a new run configuration the steps are as follows:

1. Select the project for which to create the run configuration.
2. In the Apama Studio menu bar, select Run > Run Configuration.
3. In the **Run Configuration** wizard, select Apama Application and click the New launch configuration button ().
4. On the Apama Project tab, specify the following:

- Specify the Name of the run configuration.
- Select or accept the Project.
- Select the Enable DataPlayer check box. If this box is not checked, the Query Editor and Data Player control are disabled.
- Select Generate time events from data if you want the correlator to use external time events (starting the correlator with the `-Xclock` option). The Generate time events from data check box is available only if you checked the Enable DataPlayer check box. For details about the format of correlator `&TIME` events, see *Generating events that keep time* in *Developing Apama Applications in EPL*.

  When Generate time events from data is checked, the time field specified in the playback query must be a float value that represents a number of seconds since the epoch. The data player transforms these values into &TIME events.

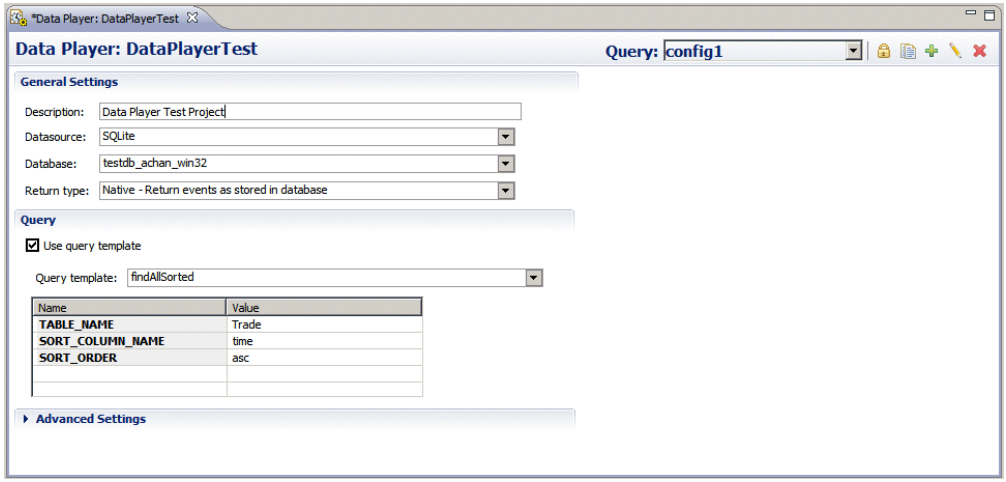
  When Generate time events from data is disabled (unchecked), the Data Player's Speed and Play to controls are disabled.

5. Click Run to save and launch the project. Or click Apply to save the configuration without running the project.

Specifying playback queries

You create and modify Data Player queries with the Data Player Query Editor. The information for the queries is stored in the project's `dataplayer_queries.xml` file. To create or modify a Data Player query:

1. In the **Project Explorer** view, expand the project and then expand the `config` folder. Double click on the `dataplayer_queries.xml` file. Apama Studio opens the Query Editor. If the project is running, you will be able to make selections from the Datasource, Database, and Query drop-down lists. If the project is not running, most of the controls are disabled.



   If the project is not running the Query Editor will report that it is offline; in this mode, you cannot select Data Sources or Databases and all the controls are disabled.

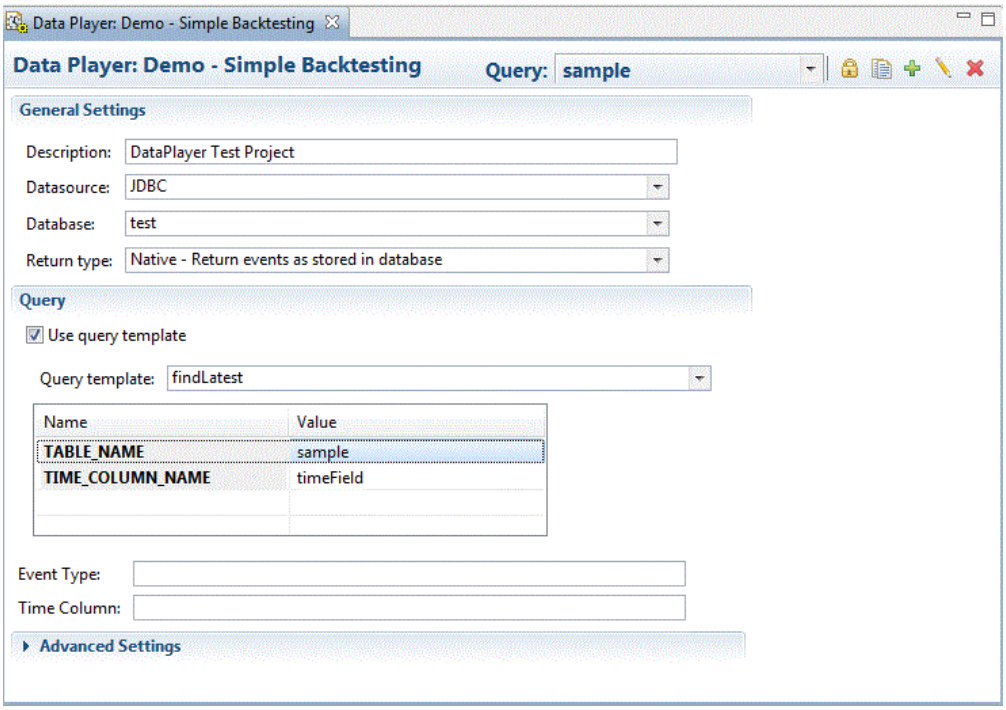
If you are creating a new query, you can also right-click on the name of the project and select New > Data Player Query from the pop-up menu. Use this method if you are working on an imported project that does not yet contain a `dataplayer_queries.xml` file.

2. Click the Add Query button( ). The **New Query** dialog is displayed.
3. Provide a name for the new query and click OK.
4. In the General Settings section, specify the following properties for the query:
   - Description — Provide a description of the query.
   - Data Source — Select an available Data Source from the drop down list.
   - Database — Select an available Database from the drop down list.
   - Return Type — The choices are `Native` or `Wrapped`. When `Native` is selected, each matching event will be sent as-is to the correlator. When `Wrapped` is selected, each matching event will be "wrapped" in a container event. The container event will be named using the event name. For example a `Tick` event would be wrapped in a `WrappedTick` event. Event wrapping allows events to be sent to the correlator without triggering application listeners. A separate user monitor can listen for wrapped events, modify the contained event, and reroute it such that application listeners can match on it.
5. In the Query section, the Use Query Template check box specifies whether you want to use a Query Template (checked) or a Raw Query (unchecked).

   If you are using a Query Template:

   a. Select a Query Template from the drop-down list. The choices are the canned queries supplied with the Apama installation: findAll, findEarliest, findLatest, and getCount. You can add your own Query Templates; see "Creating query templates" on page 166.

   b. If the Query Template uses replaceable parameters, they will be displayed in the Name and Value table in the Query section of the editor.



6. Double click a cell in the Value column and type the entry to use for the query, for example the name of a database table or column.

   If you are using a Raw Query, in the Query String text area specify the statement to execute as follows:

   - For ODBC and JDBC data sources, use SQL syntax.
   - For an Apama Sim data source, use the following keywords (keywords are case insensitive):

     `TIME` with `=`, `<`, `>`, `>=`, and `<=` comparators and the time. Use one or two `TIME` statements. If two `TIME` statements are used, only data that matches both statements will be returned.

     `INCLUDE` plus the event type to retrieve.

     `EXCLUDE` plus the event type to not retrieve.

   Examples of this syntax are:

   ```
   TIME=12345
   TIME=12345;INCLUDE=(com.apama.something)
   TIME=12345;INCLUDE=(com.apama.something);EXCLUDE=(com.apama.something.abc)
   TIME>12345;TIME<20000
   ```

7. In the Event Type field, specify unmapped-sql when the database table contains an `eventString` column that contains stringified Apama events. However, if the database table contains individual columns that are mapped to event fields in the ADBC adapter, then specify the name of the specific event in the Event Type field.

   The value that you specify in the Event Type field becomes the value for `_ADBCType` in the ADBC adapter.

8. In the Time Column field, specify the time column in your table. For example, if your table contains a `TransactionTime` column then you could specify TransactionTime in the Time Column field. Ensure that the value in the time column is a float value that represents a number of seconds since the epoch.

9. In the Advanced Settings section specify any Extra Parameters.

   a. Right click in the Name column and select Add from the pop up menu (or double click in the Name column).

   b. Specify the Name of the Extra Parameter.

   c. In the Value column, specify the value of the extra parameter.

10. Save the query.

    When you save the query, the query name is added to the **Data Player Control** view's drop down query list.

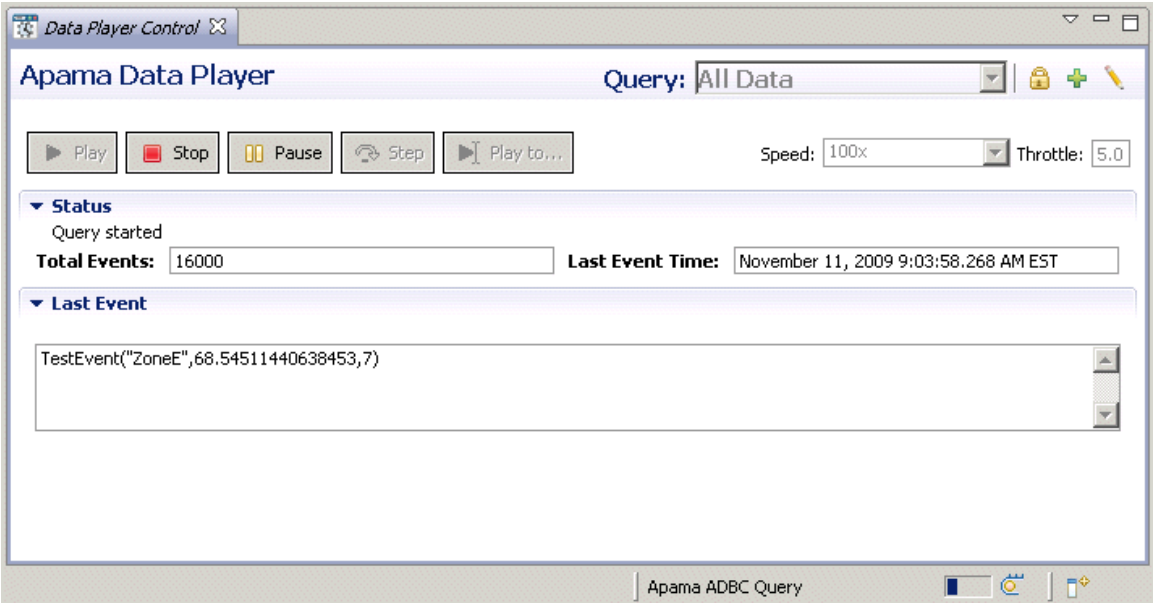
    The Query Editor tool bar contains the following buttons:

    -  Configure Credentials — This displays the **Credential for selected data source** dialog where you add a user name and password for the current query if they are required for accessing the data source and select the scope of the credentials from a drop-down list. You can specify if the credentials apply globally, to a specific data source, or to a specific query. Credentials entered here are shared by the Query Editor and the Data Player.

- Clone Query — Select this if you want to make a copy of the current query. The **Clone Query** dialog prompts you for the name of the copy. You can then edit the specifications for the query.
- Add Query — Select this if you want to add a new query. The **New Query** dialog prompts you for the name of the new query.
- Rename Query — Select this if you want to rename the query. The **Rename Query** dialog prompts for the new name of the query.
- Delete Query — Select this if you want to delete the query. The **Delete Query** dialog prompts you to confirm that this is what you want to do.

Data Player Control view

The **Data Player Control** view is enabled when the project is launched if Enable DataPlayer has been checked in the project's Run Configuration. The tab for this view is located in the lower right of the **Apama Developer** perspective next to the Console and Tasks views. If the **Data Player Control** view is not shown, select Window > Show View > Data Player Control from the Apama Studio menu.



Using the Data Player

Playback settings

The individual controls available on title bar of the **Data Player Control** view are:

- Query — The available queries are listed in the Query drop-down list on the title bar.

- Preferences — The **Preferences** dialog is displayed by clicking the view Menu down arrow at the right-hand side of the control's title bar. From the **Preferences** dialog you can select default Time Zone to use and the Date/Time Format to use during playback.
- Speed — Specify playback speed from the drop-down list; 10x is generally a good balance between speed and understandability.
- Throttle — This specifies the Data Player throttling period. The Data Player coalesces scenario update events sent to dashboards and other clients over a throttling period in order to enhance playback performance. The default setting is 5.0 seconds. You can change this setting to match your application. If data overwhelms the correlator or clients, increase the setting. If data arrives too slowly for clients to operate optimally, reduce the setting. Setting the value to 0.0 turns it off.

Data Player Control view

Playback controls

The Data Player playback controls are:

- Play — Start the playback.
- Stop — Stop the playback.
- Pause — Pause the playback.
- Step — Plays back a single event at a time.
- Play to — Displays the **Choose Date Time** dialog so you can select a specified date and time at which the playback will pause.

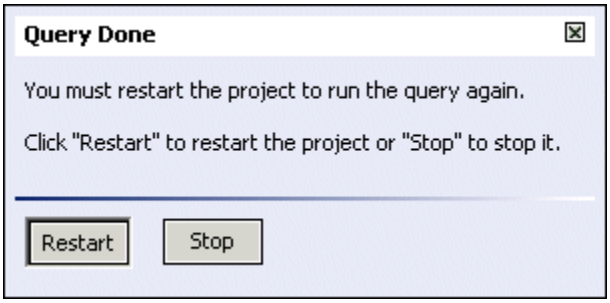
Data Player Control view

Playback status

The Data Player displays the following status information:

- Total Events — Shows the total events that have been played back during the current playback operation.
- Last Event Time — Shows the recorded time of the last event played back.
- Last Event — Shows the contents of the last event played back.

  When a playback session has run to completion or if you click the Stop button and Generate time events from data has been enabled, the Data Player control displays the **Query Done** dialog. You can move the dialog box around if it covers up status information needed in order to decide whether to restart or stop the project.



- Restart — Click Restart if you want to run another playback session. In the **Terminate launch** dialog click No. You can then display the Data Player control again and start another playback session.
- Stop — Click Stop if you want to terminate all processes associated with the playback session.

Data Player Control view

Creating query templates

If you want to create a new query template, you need to specify them in the project's query template file. The query template file is found in the project's `bundle_instance_files` folder and is named `ADBC-queryTemplates-SQL.xml` if you are using an ODBC or JDBC data source or `ADBC-queryTemplates-Sim.xml` if you are using a Sim data source. To add a query template:

1. In the **Project Explorer** view, expand the project's bundle_instance_files folder and then expand either the `ADBC_for_ODBC`, `ADBC_for_JDBC`, or `ADBC_for_Sim` folder as appropriate.
2. Double click the query templates file, either `ADBC-queryTemplates-SQL.xml` or or `ADBC-queryTemplates-Sim.xml`. The file opens in the Apama Studio text editor. The following is the standard `findLatest` query template for ODBC and JDBC data sources.

```
<query
  name="findLatest"
  description="Get the row with the latest time."
  implementationFunction="substitution"
  inputString="select * from ${TABLE_NAME} order by
         ${TIME_COLUMN_NAME} desc limit 1">
    <parameter
      description="Name of a table to query"
      name="TABLE_NAME"
      type="String"
      default=""/>
    <parameter
      description="Name of the time column"
      name="TIME_COLUMN_NAME"
      type="String"
      default="timeField"/>
</query>
```

3. The query templates are defined in the `<query>` element. The available attributes you can specify are:
   - `name` — A short name that will be displayed in the Query Template drop-down list.
   - `description` — A short description of the query.
   - `implementationFunction` — Currently query templates support the `substitution` function. This allows a token to be used as a place holder in a SQL query.

- `inputString` — For ODBC and JDBC data sources, use standard SQL syntax, from the example above:

```
inputString="select * from ${TABLE_NAME} order by
     ${TIME_COLUMN_NAME} desc limit 1">
```

In the above example, `$(TABLE_NAME)` and `$(TIME_COLUMN_NAME)` are parameters that will be replaced with a database table and column name when you create a query from the template (see `parameter`, below).

For Sim data sources use the `TIME` keyword plus the comparators `=`, `><>=`, `<=` plus the time. You can use one or two `TIME` expressions. Note, because you are adding the expression to an xml file, you need to use `>` and `<` instead of the characters `>` and `<`. Do not use spaces before or after `>` and `<`.

The `inputString` expression can also use the `INCLUDE`, and `EXCLUDE` keywords, for example:

```
TIME=12345;INCLUDE=(com.apama.something);EXCLUDE=(com.apama.something.abc)
TIME&gt;12345;TIME&lt;20000
```

- `parameter` — For ODBC and JDBC data sources you can specify parameters and default values for the query template. This allows you to create different queries from the same template that, for example, query different tables or different columns in the database. From the above example:

```
<parameter
  description="Name of a table to query"
  name="TABLE_NAME"
  type="String"
  default=""/>
<parameter
  description="Name of the time column"
  name="TIME_COLUMN_NAME"
  type="String"
  default="timeField"/>
```

When you create a query from a template that has the parameters shown above in the Query Editor, you will specifiy the names of the database table and column to use with the query.

4. Save the query template file when you finish modifying it.

Using the Data Player

Command-line Data Player interface

When you are ready to test your application, you can use the Data Player command-line interface to write scripts and unit tests to exercise the API layers. In some cases it might be easier to play back events to the correlator using the command-line interface as compared to using the Data Player GUI in Apama Studio.

To use the command-line interface to the Data Player, you must have already used the GUI interface in Apama Studio to define queries and query configurations in Apama Studio. When you use the command-line interface, you specify the query names and query configurations that you created in Apama Studio.

For more information on using the Data Player command line interface, see "Using the Data Player command line interface" in *Deploying and Managing Apama Applications*.

Using the Data Player

Chapter 9: Generating Dashboards

This section describes how to generate dashboards for a given scenario by using the **Dashboard Generation** wizard.

Dashboards provide the ability to view and interact with scenarios and DataViews. They contain charts and other objects that dynamically visualize the values of scenario variables. Dashboards can also contain control objects for creating, editing, and deleting scenario instances. You create Dashboards either with the **Dashboard Generation** wizard or with the Dashboard Builder.

The wizard allows you to generate simple, default dashboards, customized by your choices regarding layout, visualization objects to display, and scenario variables to use with each visualization object.

The Builder is a graphical composition tool that gives you fine-grained control over a dashboard's appearance and behavior. It also supports a wider array of visualization and control objects than does the wizard. Advanced users can use the Builder instead of the wizard to create dashboards from scratch, or they can use the Builder in conjunction with the wizard to modify or augment generated dashboards. See *Building Dashboards* for more information on the Dashboard Builder.

The wizard allows you to do all the following:

- Create and edit dashboard-generation configurations
- Save configurations to an XML file
- Generate dashboards from a configuration

Once you have finished generating your dashboards with the wizard (or finished building or modifying them with the Builder), follow the steps described in "Preparing Dashboards for Deployment" on page 177.

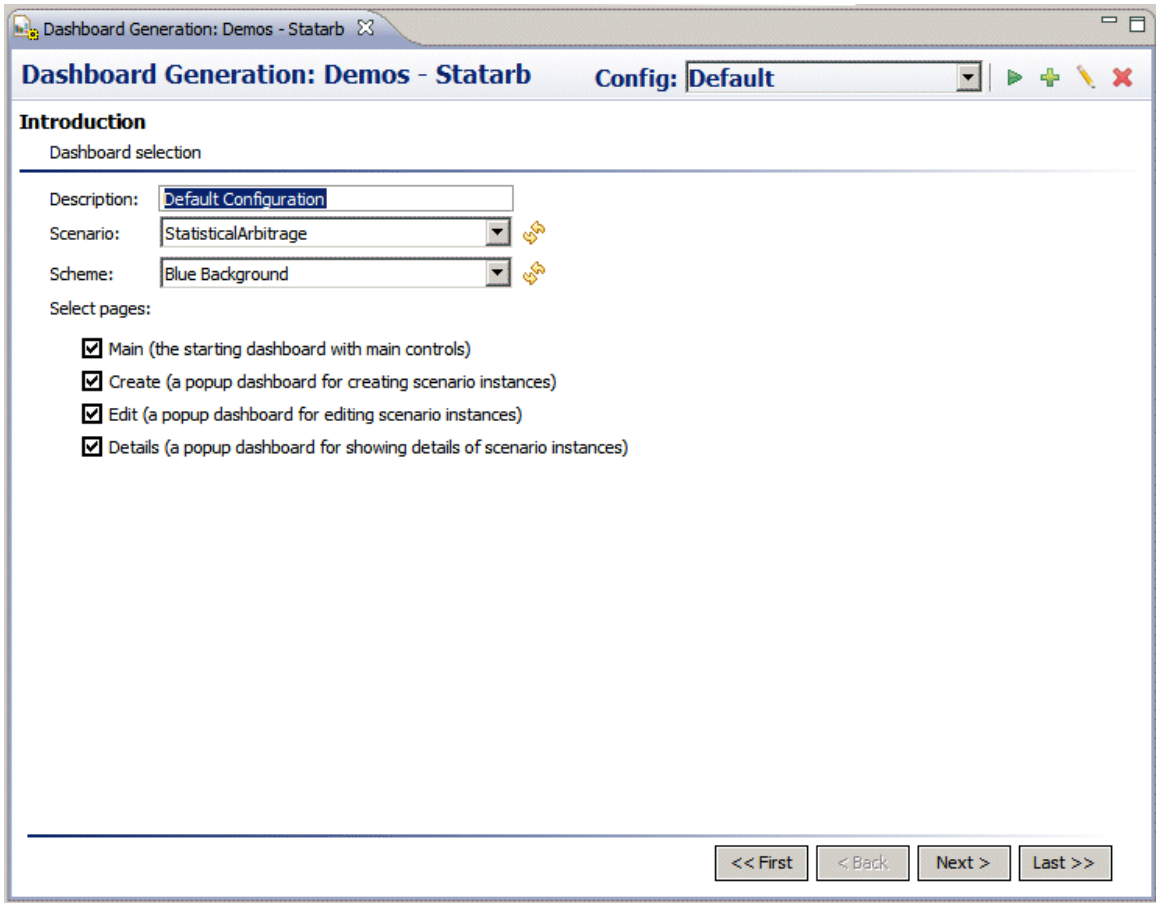
Note that if you modify a dashboard with the Builder, the changes you make cannot be propagated back to the configuration that generated the dashboard. So once you modify a dashboard with the Builder, you no longer use the wizard for development of that dashboard.

Starting the wizard

You start the **Dashboard Generation** wizard by opening your project's dashboard-generation configuration file, `dashboard_generation.xml` in your project's `config` folder. This is the file in which saved configurations are stored.

Follow these steps to start the wizard:

1. Open your project's `config` folder. If `config` or `config\dashboard_generation.xml` does not exist (because your project was created with a prior Apama release), right click on the project folder and select Add Dashboard Generation Configuration from the pop-up menu.
2. Double-click on `dashboard_generation.xml`, or else right click on it and select Open With > Apama Dashboard Generation Editor from the pop-up menu. The wizard appears.



Generating Dashboards

Using the wizard

Use of the wizard involves the following steps:

1. Use the toolbar to create a new dashboard-generation configuration, or to select a previously saved configuration. See "Using the titlebar/toolbar" on page 170.
2. For each of the wizard's forms, fill out or modify the settings (or accept the default or previously saved settings), and then click Next.
3. On the toolbar, click the  tool icon in order to generate the dashboards. See "Using the titlebar/toolbar" on page 170.

At any time, you can save configuration changes by selecting Save in the Apama Studio File menu.

The wizard's forms are discussed in the following sections:

- "Using the Introduction form" on page 170
- "Using the Main, Create, Edit, and Details Forms" on page 171
- "Using the layout configuration forms" on page 173

Generating Dashboards

Using the titlebar/toolbar

The titlebar/toolbar is located at the top of the wizard. It allows you to select a configuration to edit. It also allows you to add, remove, and rename configurations.

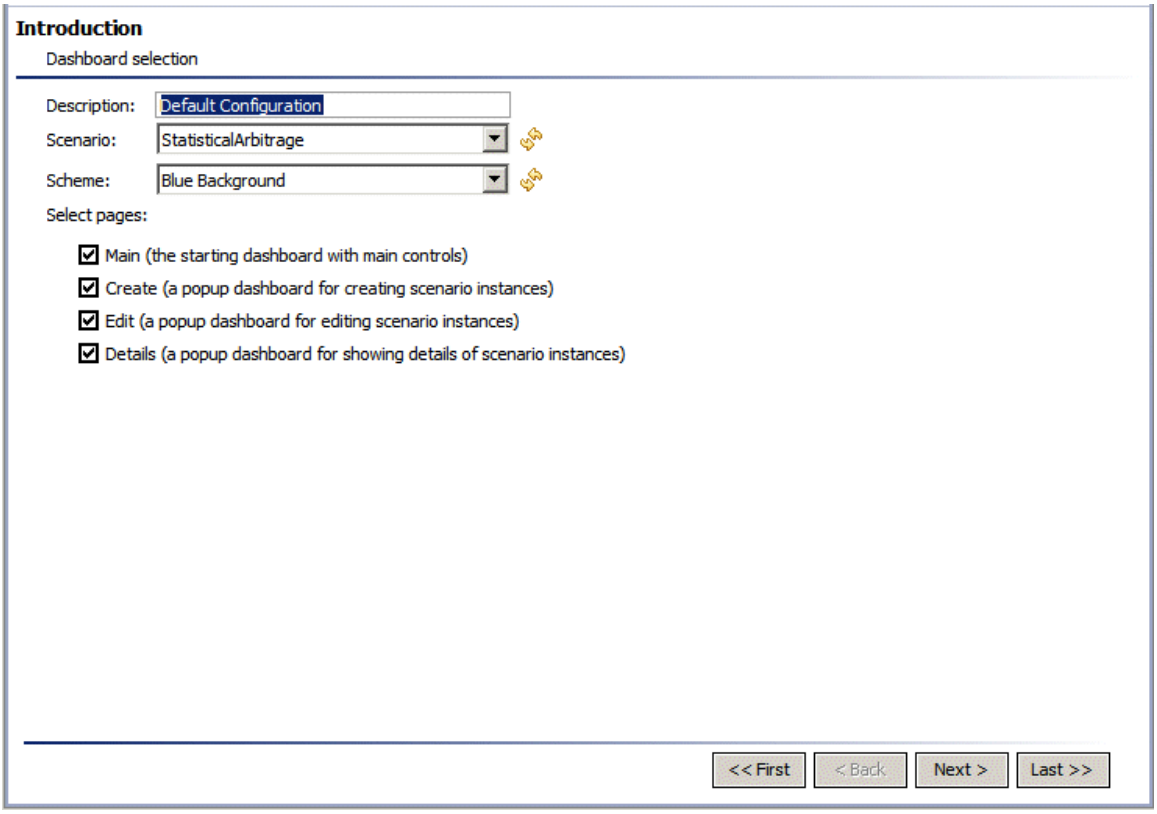
The titlebar/toolbar includes the following elements:

- Title: At the far left, Dashboard Generation: followed by the project name appears.
- Config field: A drop down list of configurations appears next to the label Config:. These are all the existing configurations for the current project. When you edit a configuration, or generate dashboards for a configuration, you must first select the configuration from the list.
-  tool icon: Generates dashboards for the configuration specified in the Config field (see above), and displays a dialog that indicates the location of the generated dashboards. Generated dashboards are placed in your project's `dashboards` folder.
-  tool icon: Adds a new, named configuration
-  tool icon: Renames the configuration specified in the Config field (see above).
-  tool icon: Removes the configuration specified in the Config field (see above).

Generating Dashboards

Using the Introduction form

The **Introduction** form appears when the wizard first starts, as well as when you add a new configuration. It allows you to specify a text description for the dashboards, a scenario for which to generate the dashboards, and a color scheme for the dashboards. It also allows you to specify the types of dashboards to generate.



The Introduction form has the following editable components:

- Description: Enter an optional text description of the current dashboard-generation configuration.
- Scenario: Select the scenario for which dashboards are to be generated. The visualization objects of the generated dashboards are attached to data from the specified scenario's variables.

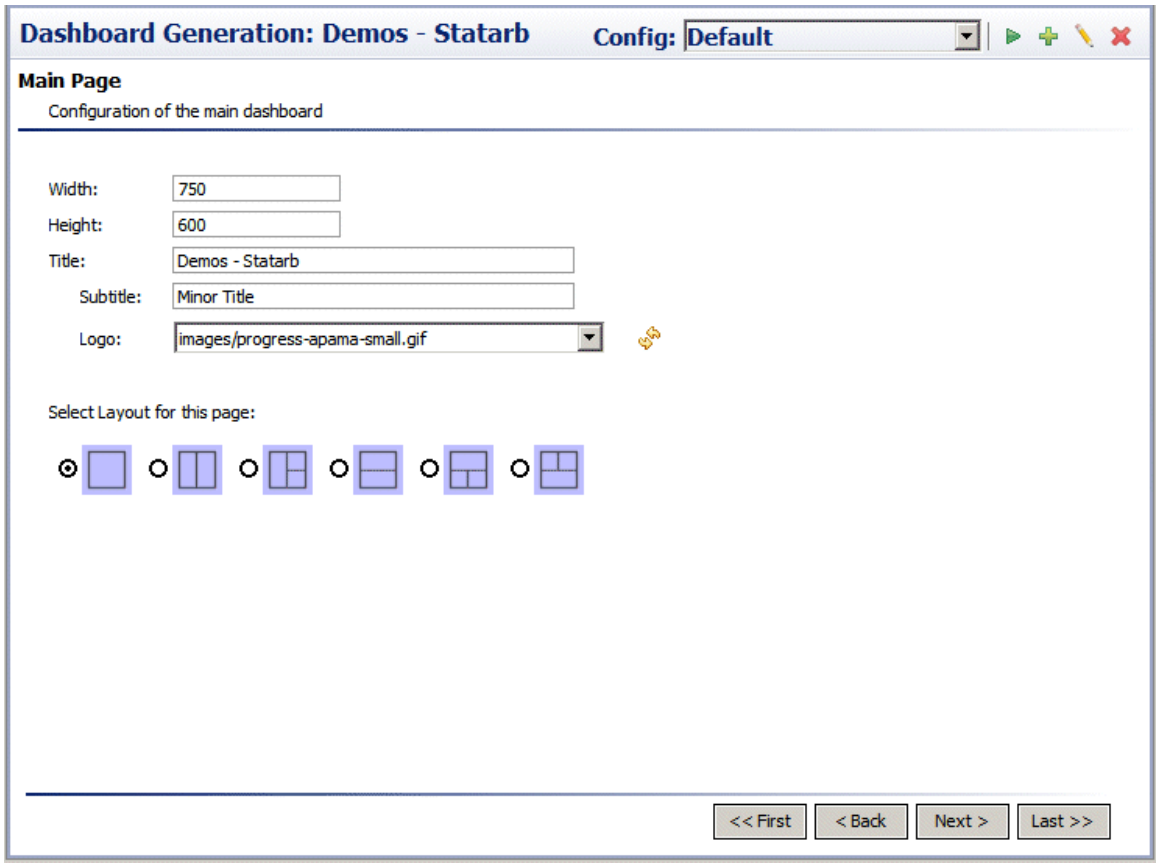
  This dropdown menu is disabled if your project has no scenarios. After you import or create a scenario, click on the icon next to the dropdown menu in order to refresh its contents.
- Scheme: Select an item from the dropdown list in order to control either the dashboard background color or the fill color of visualization objects and forms in the dashboards.
- Select pages: Click the checkbox for each type of dashboard that you want to be generated.
- Main: First dashboard page that the end user sees in each session. Contains buttons that allow the end user to open other dashboards.
- Create: Allows end users to create new scenario instances.
- Edit: Allows end users to edit the selected scenario instance.
- Details: Provides a detailed view of scenario instances or of a selected scenario instance.

Generating Dashboards

Using the Main, Create, Edit, and Details Forms

These forms allow you to specify information about each dashboard to be generated (as specified in the Select pages section of the Introduction form), including height, width, titles, logo, and layout.

One of these forms appears when you press Next in the Introduction form, and when you press Next in the layout configuration form for the previous dashboard page (main, create, or edit).



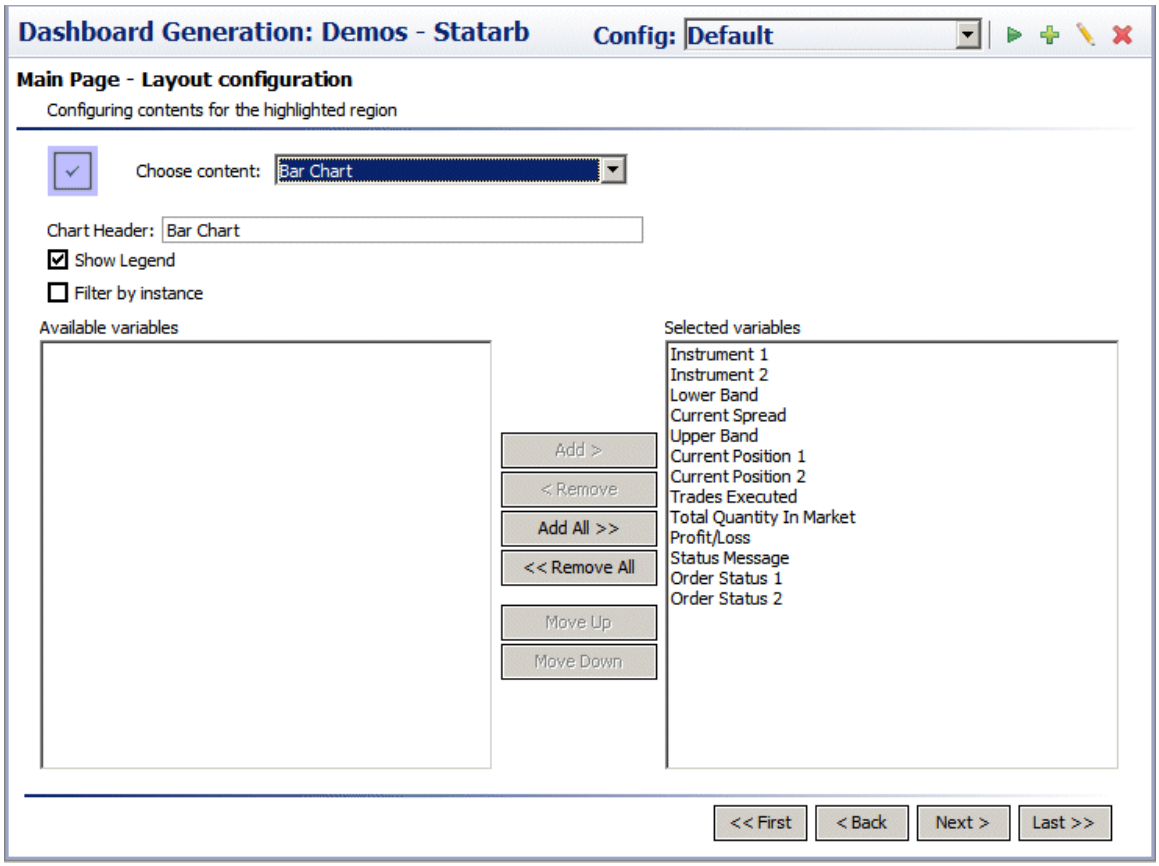
For each dashboard, the wizard includes a form with the following editable components:

- Width: Specify the width of the dashboard window in pixels.
- Height: Specify the height of the dashboard window in pixels.
- Title: Enter the title of the dashboard window. The title appears in the top, left portion of the window. This field is optional; you can leave it empty.
- Subtitle: Enter the subtitle of the dashboard window. The subtitle appears beneath the Title. This field is optional; you can leave it empty.
- Logo: Select a graphic file. The dropdown list includes all supported graphic files in the dashboard \images folder under your project folder. Supported formats include GIF, JPG, and PNG.The logo appears in the top, right portion of the generated dashboard. After you import or create a new graphic file, click on the icon next to the dropdown menu in order to refresh its contents. This field is optional; you can leave it empty.
- Layout: Click the radio button for the desired layout. Each section of a dashboard's layout contains one visualization object (table, bar graph, or pie chart) or one form, as specified in the layout configuration forms.

Generating Dashboards

Using the layout configuration forms

These forms allow you to enter information about each section of each dashboard's layout, including the visualization object to use in that section, and how to attach the object to scenario variables.



For each section of each dashboard (main, create, edit, or details), the wizard includes a form with the following editable elements:

- Choose content: Select an object to appear in the current section of the layout. The current section of the layout is indicated by the check mark in the layout diagram to the left of this field.
- Scenario variables table: Use the buttons to ensure that the Selected variables column contains those variables that you want attached to the object specified in the Choose content field. Select an item or items in the Available variables column to activate the Add button; select an item or items in the Selected variables column to activate the Remove button.

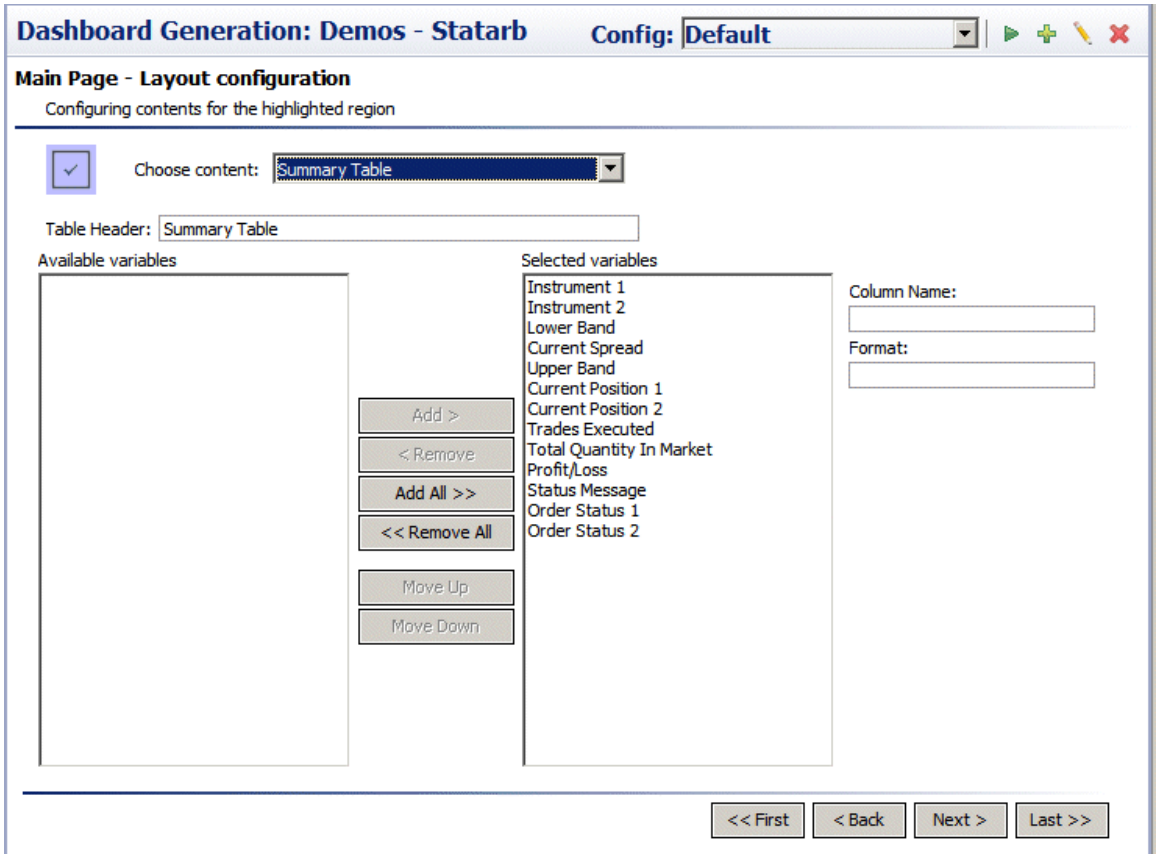
Summary tables contain one row for each scenario instance, and one column for each variable that is included in the Selected variables column. A cell in a given column and row contains the value of the column's corresponding variable for the row's corresponding scenario instance.

Bar charts contain one group of bars for each numeric variable that is included in the Selected variables column. Within each group, there is one bar for each scenario instance. The size of a given bar in a given group is proportional to the value of the group's corresponding numeric variable for the bar's corresponding scenario instance.

Pie charts contain one slice for each scenario instance. The size of a given slice is proportional to the value of the first included, numeric variable for the slice's corresponding scenario instance.

Form panels contain one text-entry field for each scenario variable that is included in the Selected variables column. Create and Edit dashboards use entered values in order to initialize or update the variables.

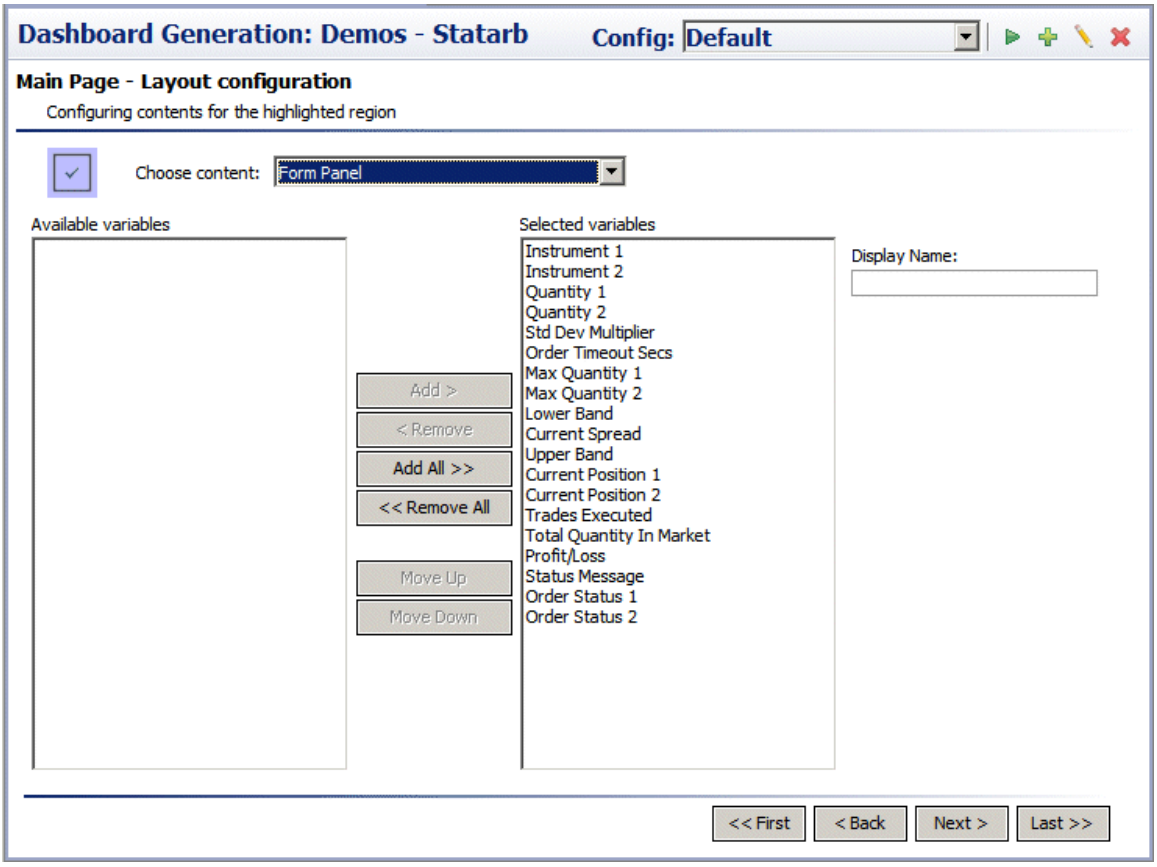
If you select Summary Table for the Choose content field, the following elements are also included:



- Table header: Enter a label for the table
- Column name: Select a variable in the Selected variables column of the scenario variables table. Enter the header for the selected variable's corresponding table column. Leave this field blank to use the variable name as the column header.
- Format: Select a variable in the Selected variables column of the scenario variables table. Enter a format string for the selected variable's corresponding table column. Specify numerical formats based on the Java format specification, or with the following shorthand:
- $ for US dollar money values
- $$ for US dollar money values with additional formatting, ( ) for non-money values, formatted similar to money
- # for positive or negative whole values

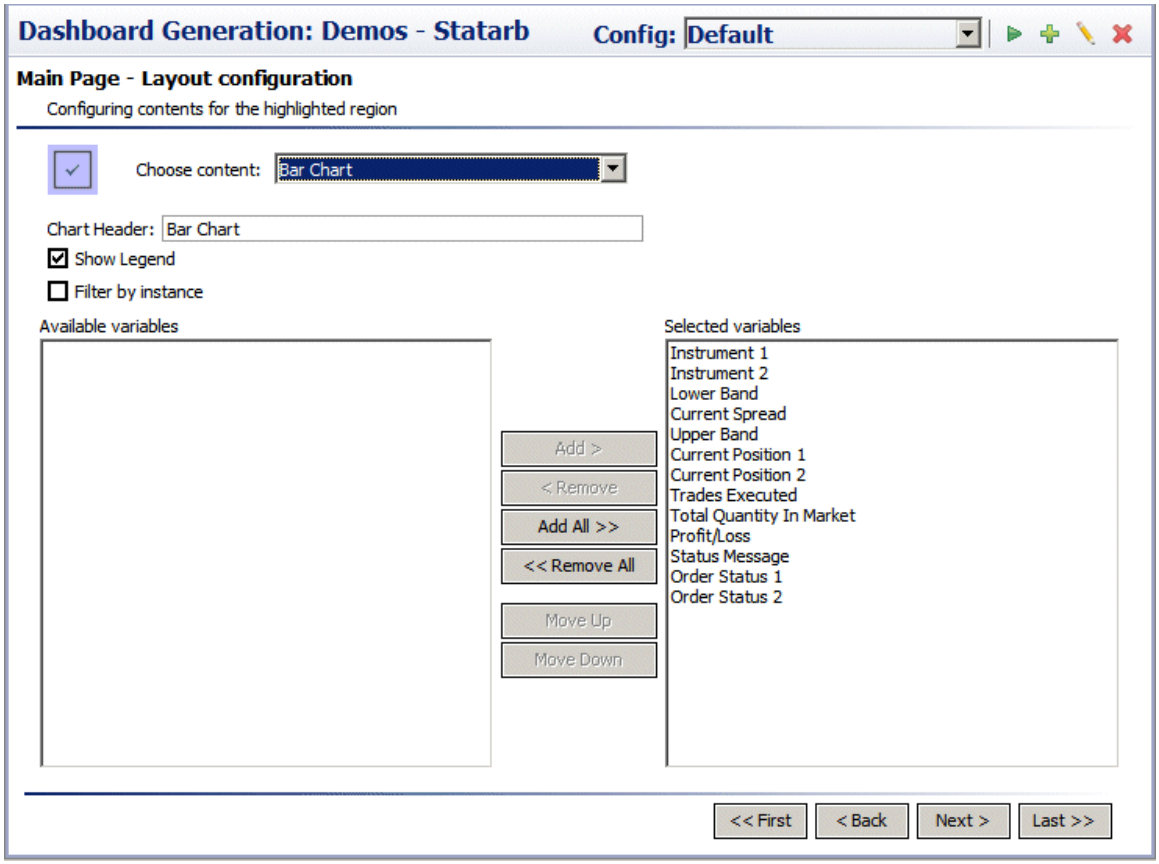
Specify date formats based on the Java date specification.

If you select Form Panel for the Choose content field, the following element is also included:



- Display name: Select a variable in the Selected variables column of the scenario variables table. Enter a label for the selected variable's corresponding text-entry field. Leave the Display name field blank to use the variable name as the field label.

If you select Bar Chart or Pie Chart for the Choose content field, the following elements are also included:



- Chart header: Enter a heading for the chart.
- Show legend: Select to show a legend for the chart. The legend indicates the mapping from bar or pie-slice color to value of the scenario's first non-numeric variable for the bar or slice's corresponding scenario instance.
- Filter by instance: Select to filter out all scenario instances except the one that corresponds to the selected row in a summary table on the current dashboard. This allows the end user (the user of the generated dashboard) to select the scenario instance to be visualized. In this case, the bar chart has only a single bar for each numeric variable, rather than a group of bars for each numeric variable.

Generating Dashboards

Chapter 10: Preparing Dashboards for Deployment

This section describes how to prepare a project's dashboards for deployment, including how to create a deployment configuration with the Dashboard Deployment Configuration Editor, as well as how to use the Packaging wizard to generate a deployment package.

Once you have followed the steps described here, if you want to deploy on additional application servers without using Apama Studio, you or another user must follow the steps described in *Deploying and Managing Apama Applications*, "Deploying Dashboards".

Follow these steps in order to prepare a project's dashboards for deployment, generate a deployment, packag:

1. Ensure that the dashboards have the required functionality. See "Dashboard feature checklist" on page 177.
2. Change your dashboard's correlator definitions so that they specify deployment correlators. See "Changing correlator definitions for deployment" on page 178.
3. Decide which type or types of deployment to support for your project. See "Choosing among deployment types" on page 178.
4. Create a deployment configuration or deployment configurations by using the Dashboard Deployment Configuration Editor. See "Using the Deployment Configuration editor" on page 181.
5. Generate a deployment package either with the Dashboard Package wizard or with the `dashboard_management` command line tool. See "Using the Dashboard Package wizard" on page 185 and "Generating a deployment package from the command line" on page 186.
6. If necessary, communicate the appropriate information to the individual who will complete the deployment process. See "Sharing information with the Dashboard Administrator" on page 187.

Dashboard feature checklist

This section contains a checklist of capabilities that you should include in a project's dashboards in order to ensure that the dashboards provide all standard dashboard functions. Most projects require all these capabilities, but some projects may not.

- Summary view: Displays a listing of all the instances of a scenario or all the items of a DataView.
- Detail view: Provides detailed information about a selected scenario instance or DataView item.

- Create: Allows creation of new scenario instances or DataView items.
- Edit: Supports editing of existing scenario instances or DataView items.
- Delete: Allows deletion of scenario instances or DataView items.

The Statistical Arbitrage sample included with Apama is an example of a scenario dashboard that provides all these capabilities.

Preparing Dashboards for Deployment

Changing correlator definitions for deployment

When you create a dashboard in Dashboard Builder, use a development correlator. When you deploy a dashboard for use with a live correlator, change the correlator host and port so that they reference the live correlator.

This can be done in two ways:

- In the Dashboard Builder, select Tools > Options... and use the Apama tab to specify the deployment correlator or correlators. You must do this *before* you generate a deployment package with the Dashboard Deployment Configuration Editor. See Specifying Data Sources on page 29 in *Building Dashboards*.
- When you or another user starts a Data Server or Display Server that will serve event data to your deployed dashboard, use the `-c` or `--correlator` option to override the host and port specified in Dashboard Builder for a given correlator logical name. See Managing the Dashboard Data Server and Display Server in *Deploying and Managing Apama Applications*.

If a user other than you will complete the deployment as described in *Deploying and Managing Apama Applications*, you must communicate to this other user the logical name for each correlator as well as the host name and port for each *deployment* correlator (if any) that you defined.

Preparing Dashboards for Deployment

Choosing among deployment types

Apama supports two types of dashboard deployment:

- Web-based: as an applet or Java Web Start application, or as a simple, thin-client Web page (thin-client deployment is known as *Display Server deployment*, because it uses the Display Server to mediate correlator access)
- Local: as a locally-installed desktop application (the Dashboard Viewer) together with dashboard-specific files that the application can open

The following sections compare Web-based deployments with local deployments with regard to these factors:

- "Application installation" on page 179
- "Authentication" on page 179
- "Authorization" on page 179

- "Data Protection" on page 180
- "Scalability" on page 180

They are followed by a section on "Choosing among Web-based deployment types" on page 180.

Note, the valueHigh(Low)AlarmCommand property of Range Dynamic Objects only works for non-display server deployments. For display server deployments, only the valueHigh(Low)AlarmImage and the valueHigh(Low)Color properties will be honored.

Preparing Dashboards for Deployment

Application installation

Local deployments require the use of the Dashboard Viewer desktop application (available on Windows platforms only). End users open locally-deployed dashboards in the Dashboard Viewer, which must be pre-installed locally or on a shared file system. See the *Using the Dashboard Viewer* for more information.

With Web-based deployment, the Dashboard Viewer does not need to be installed locally. Dashboards are invoked through a Web browser, and are installed on demand, as simple web pages, applets, or Web Start applications, so they can easily be deployed across a wide area network, including the Internet.

Authentication

Web-based deployments provide Web-based login functionality and use the authentication mechanism provided by your application server. They support authentication customization by allowing you to configure your application server to use the security realm and authentication service of your choice.

Local deployments include Data Server or Display Server login functionality, and support authentication by allowing you to supply any JAAS-supported authentication module as a plug-in to the Data Server or Display Server.

Authorization

Web-based deployments support role-based dashboard access control, which allows you to associate a role with a deployed dashboard, and to authorize use of the dashboard only for application server users with the dashboard's associated role.

Local deployments support dashboard access control by allowing you to use the system security mechanisms in order to restrict access to the deployed dashboard files.

Both types of deployment support scenario and DataView access control, which allows you to control who can have which type of access to which scenarios and DataViews.

Data Protection

With Web-based deployments you can secure inter-process communication by enabling HTTPS in the application server. With local deployments you can secure inter-process communication by enabling secure sockets (SSL) in the Data Server or Display Server.

With both types of deployment you can secure inter-process communication through the use of secure channels (SSH) and virtual private networks (VPN).

Scalability

Both types of deployment are highly scalable, since both use the Data Sever or Display Server to mediate access to correlators.

Choosing among Web-based deployment types

Each Web-based deployment is one of the following:

- Applet
- WebStart application
- Thin-client Web page

The following sections compare these three types of deployment with regard to the following factors:

- "Installation" on page 180
- "Served data" on page 180
- "Refresh latency" on page 181
- "Dashboard command support" on page 181

Installation

For applet and WebStart deployments, the local Web browser must have the Java plug-in. For thin-client Web page deployments, no Java plug-in is required.

Served data

The Data Server mediates correlator access for applet, WebStart, and local deployments; the Display Server mediates correlator access for simple thin-client, Web-page deployments.

The Data Server delivers raw data from which deployed dashboards construct the visualization objects that they display. The Display Server, in contrast, delivers already-constructed visualization objects in the form of image files and image maps, and therefore no Java plug-in is required on clients of the Display Server.

Refresh latency

The Display Server's minimum refresh latency (5 seconds) is greater than that of the Data Server. Use the Data Server for applications that require high-frequency screen updates.

Dashboard command support

Applet deployments do not support dashboard actions that are system commands (that is, actions that are specified with the **Define System Command** dialog).

Dashboard iPad Support

Dashboards targeted for the Apple iPad must be Display Server deployments.

Using the Deployment Configuration editor

The Deployment Configuration Editor is a form-based interface that allows you to specify dashboard deployment configurations and save them for future use. It also allows you to launch the Packaging wizard, which you can use to generate deployment packages (`.war` files or `.zip` files) from configurations.

See the following sections for detailed information:

- "Starting the Configuration editor" on page 181
- "Saving deployment configurations" on page 182
- "Sections of the configuration editor GUI" on page 182
- "Title bar/Toolbar" on page 182
- "General Settings" on page 183
- "Layout" on page 185
- "Additional JAR Files" on page 184
- "Startup and Server" on page 184
- "Using the Dashboard Package wizard" on page 185

Preparing Dashboards for Deployment

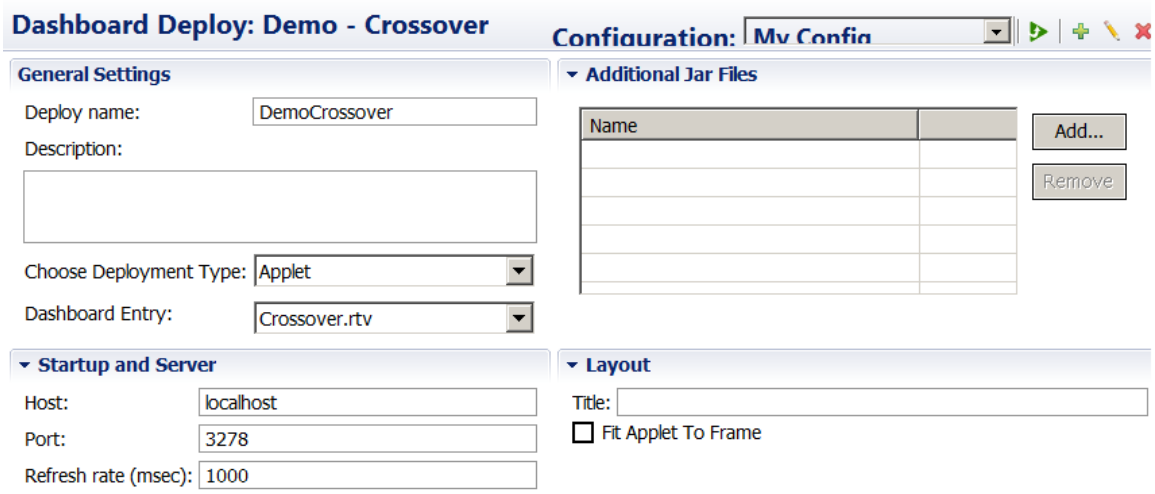
Starting the Configuration editor

Follow these steps to start the Configuration Editor:

1. If you are using the **Project Explorer** view, ensure that a project is selected.
2. In either the **Project Explorer** view or the **Workbench Project** view, select New > Dashboard Deployment from the File menu. (You can also right-click in the navigation pane and select Dashboard Deployment from the popup menu. In the **Workbench Project** view, you can also click the New button

that is above the navigation pane and select Dashboard Deployment from the Apama folder, or click the down arrow that is next to the New button, and select Dashboard Deployment from the popup menu.)

3. In the **New Dashboard Deployment Configuration** dialog, enter a name in the Configuration field. The Dashboard Deployment Configuration Editor uses this as the name of the new configuration.
4. Click Finish.The Dashboard Deployment Configuration Editor appears, and displays the new dashboard configuration. In addition, a new dashboard-deployment configuration file (`dashboard_deploy.xml`) appears under the current project's config folder, if one wasn't already present.



Saving deployment configurations

Select Save in the Apama Studio File menu to save configuration changes.

Sections of the configuration editor GUI

The Configuration Editor has the following sections:

- "Title bar/Toolbar" on page 182
- "General Settings" on page 183
- "Startup and Server" on page 184
- "Additional JAR Files" on page 184
- "Layout" on page 185

Title bar/Toolbar

The title bar/toolbar appears at the top of the Configuration Editor. It allows you to select a configuration to edit. It also allows you to add, remove, and rename configurations, as well as to start the Packaging wizard.

The title bar/toolbar includes the following elements:

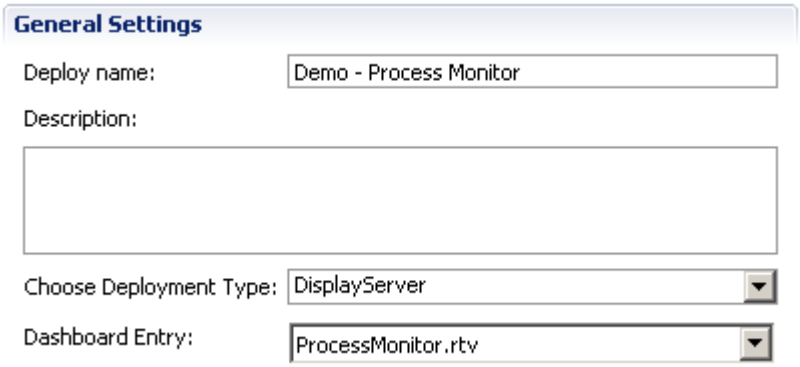
- Title: Dashboard Deploy: followed by the project name appears at the far left.
- Configuration field: A drop down list of configurations appears next to the label Configuration:. These are all the existing dashboard configurations for the current project. When you edit a configuration, you must first select the configuration from the list.
-  Dashboard Package: Starts a wizard that can generate deployment packages for one or more of the available configurations.

See "Using the Dashboard Package wizard" on page 185.

-  Add: Adds a new, named configuration
-  Rename: Renames the configuration specified in the Configurations field (see above).
-  Remove: Removes the configuration specified in the Configurations field (see above).

General Settings

This section allows you to specify a name, description and deployment type, as well as an entry-point dashboard file or a panels configuration file.



It has the following editable elements:

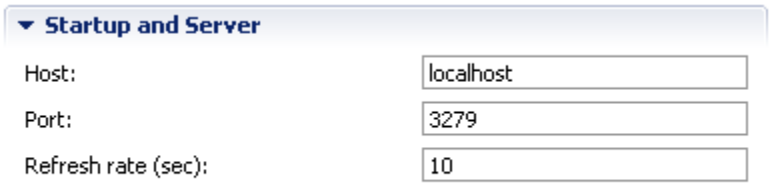
- Deploy Name text field: Enter a name to be used as the file name of the generated deployment package. This name is also used as the directory name for temporary deployment files. Do not use spaces in this field.
- Choose Deployment Type: Select the type of deployment for which you want to prepare your dashboard: WebStart, Applet, Display Server, or Local.

- Dashboard Entry: Select the dashboard entry point, the file to be used as the initially-displayed dashboard. If you are using multiple display panels, select a panels-initialization file.

If a user other than you will complete the deployment as described in *Deploying and Managing Apama Applications*, you must communicate to this other user the file name specified in the Deploy Name text field as well as the deployment type chosen from the Choose Deployment Type drop down list.

Startup and Server

If you selected a Web-based deployment type (that is, Web Start, Applet, or Display Server deployment), the Startup and Server section is visible:



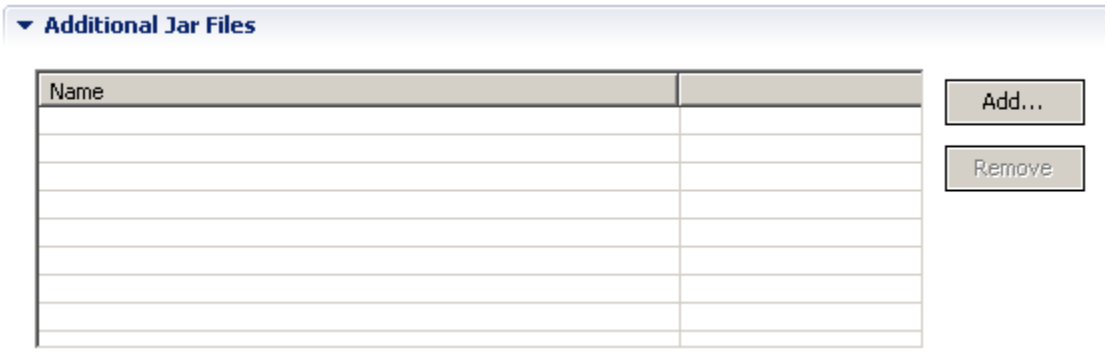
This section contains the following editable elements:

- Host text field: Specify the host of the Data Server or Display Server that will serve data to the deployed dashboard.
- Port text field: Specify the port of the Data Server or Display Server that will serve data to the deployed dashboard.
- Refresh Rate text field: Specify the dashboard update rate, which is the rate at which the dashboard updates its display to reflect new event data received from the correlator via the Data Server or Display Server. If you know the maximum update rate used by the Servers to which the deployed dashboard might connect, ensure that the update rate that you specify here is no greater than this maximum.

If a user other than you will complete the deployment as described in *Deploying and Managing Apama Applications*, you must communicate to this other user the host, port, and refresh rate that you specified.

Additional JAR Files

For backward compatibility, this section allows you to specify additional `.jar` files. These additional files must be in the directory `%APAMA_WORK%\dashboards_deploy\lib`.



The JAR files table is for backward compatibility only; specify new `.jar` files in the Apama Studio Dashboard Properties (select Properties from the Project menu). Click the Add button to display a list of `.jar` files that are found in the directory `%APAMA_WORK%\dashboards_deploy\lib`. Select the file or files that you want to add. To remove files that have been added, select them from the table and click Remove.

Layout

This section allows you to control the appearance of Applet/WebStart deployments. It contains the following editable elements:

- For Applet/WebStart deployments, you can supply a title that will be used by the deployed application or applet. This is optional.
- For Applet deployments, if Fit Applet To Frame is checked, the applet's height and width are set to 100%.

Using the Dashboard Package wizard

The Dashboard Package tool,  (which is on the "Title bar/Toolbar" on page 182) brings up a wizard that guides you through the process of generating a deployment package for specified configurations.

Follow these steps to use the wizard:

1. Click The Dashboard Package tool  (which is on the "Title bar/Toolbar" on page 182) to start the wizard.

   The Dashboard Selection screen appears.

2. In the Available Configurations section, use the check boxes to select one or more configurations on which to perform the specified operations and click Next. The **Package Dashboard Configurations** appears.

   The Package Dashboard Configurations screen appears.

3. In the Package Location field, enter or browse to the path name of the directory into which you want Apama to place the generated deployment package. For local deployments, this is a directory that is accessible to end users.

   If the desired final destination of the deployment package is not accessible to you, the deployment package can be installed by the dashboard administrator as part of the deployment process. See Deploying Dashboards, in *Deploying and Managing Apama Applications*.

4. Your deployment can include `.jar` files that define custom classes and functions used by your project's dashboards. The `.jar` files that are specified in Dashboard Properties (select Properties from Apama Studio's Project menu) are automatically included in the generated deployment package. This screen allows you to direct Apama to sign `.jar` files before including them in the deployment package.

   Click the Default button, to specify the keystore shipped with Apama (`%APAMA_HOME%\etc\DashboardKeystore`). Use the default unless you require a custom keystore. If you require a custom keystore, use the following fields:

   - Signature file text field: Enter the full path name of (or click Browse... and navigate to) the keystore to use for singing `.jar` files. Leave this field empty to skip signing the `.jar` files.
   - Alias text field: Enter the private key to be used to sign the `.jar` files. If you are using the keystore shipped with Apama, click the Default button (which specifies the alias `dashboard`).
   - Password text field: Enter the password for the private key specified in the Alias text field. Or click the default button for default Alias/Password being used for the DashboardKeystore.

5. Click Finish.

   The operations are performed and then the Dashboard package/deploy/publish summary appears. The summary indicates which operations succeeded for which configurations. A green check mark indicates success. A red x indicates failure.

Generating a deployment package from the command line

Once you have a defined a dashboard deployment configuration, use `dashboard_management` in order to generate a deployment package. Use the following options:

- `-y` or `--deploy`: Specify a dashboard configuration file, typically `dashboard_deploy.xml` in your project's `config` folder.
- `-c` or `--config`: Specify the name of a deployment configuration that is saved in the file specified with `-y` or `--deploy`.
- `-r` or `--rtvPath`: Specify the directory containing the dashboard (`.rtv` files) to use in order to generate the deployment package.
- `-k` or `--keystoreFile`: Specify the keystore to use in order to sign the `.jar` files to be included in the deployment package. Supply this option only if the `.jar` files are not already signed.
- `-a` or `--alias`: Specify the alias to use in order to sign the `.jar` files to be included in the deployment package.
- `-j` or `--jar`: Specify a third-party jar file to sign. You can specify multiple `-j | --jar` arguments if you have multiple jar files to sign.

- `-o` or `--password`: Specify the password to use in order to sign the `.jar` files to be included in the deployment package.

Here is an example:

```
dashboard_management --deploy "C:\workspace\Demo - Statistical
Arbitrage\config\dashboard_deploy.xml" --config "My Config" --rtvPath "C:\workspace\Demo
- Statistical Arbitrage\dashboards" --keystoreFile "C:\Program Files\Software AG\Apama
5.2\etc\DashboardKeystore" --alias "dashboard" --password "terra"
```

For more information on `dashboard_management`, see Managing and Stopping the Data Server and Display Server in *Deploying and Managing Apama Applications*.

Preparing Dashboards for Deployment

Sharing information with the Dashboard Administrator

There are two types of activity involved in making dashboards available to end users:

- Dashboard development, which requires the use of the Apama Dashboard Builder or **Dashboard Generation** wizard, as well as the use of the Dashboard Deployment Configuration Editor to generate a deployment package.
- Dashboard deployment, which requires installing and configuring the deployment package, as well as administering the Data Server or Display Server and managing dashboard security (see *Deploying and Managing Apama Applications*).

Sometimes these activities are performed by different individuals. In such a case, the dashboard developer must be sure to communicate the following information to the dashboard administrator regarding the dashboard to be deployed:

- Location and file name of the `.war` file or `.zip` file that was generated by the Deployment Configuration Editor when the developer prepared the dashboard for deployment
- For Display Server deployments, the location of the dashboard project directory (the directory that contains the project's `.rtv` files)
- For Web-based deployments, the Data Sever or Display Server host, port, and update rate that the builder supplied to the Configuration Editor
- Logical name for each correlator as well as the host name and port for each *deployment* correlator (if any) that was specified by the dashboard developer in the Apama tab of the Tools > Options... dialog prior to the generation of the deployment package. See "Changing correlator definitions for deployment" on page 178.
- Trend-data caching requirements for the deployed dashboards. See Configuring Trend-Data Caching in *Deploying and Managing Apama Applications*.
- Whether SQL-based instance tables are used by the dashboard for data attachments. See Attaching dashboards to correlator data in Building dashboards in *Developing Apama Applications*.

Preparing Dashboards for Deployment

Chapter 11: File Definition Formats

This section describes the formats of Apama's function definition and block definition files. It is important that developers adhere strictly to these formats when developing functions and blocks to be used in Apama scenarios.

Understanding the XML format is especially important for developers creating functions, because the function editor in Apama Studio lets you work directly on a function definition file's XML code. Function definition files have an `.fdf` extension.

On the other hand, block developers are shielded from most of a block definition file's XML code by the Apama Studio block editor, which automatically generates the block's boilerplate code and allows input only in sections of the file where user input is appropriate. Block definition files have a `.bdf` extension.

Function definition file format

A function definition file contains metadata that describes the function plus EPL code that implements the function. The following topics describe these pieces:

- "Defining metadata in function definition files" on page 188
- "Defining EPL code in function definition files" on page 191

File Definition Formats

Defining metadata in function definition files

The metadata in a function definition file has the following format:

```
<function name="string" display-string="string" return-type="string">
   <version>
      <id>version_number</id>
      <date>version_date</date>
      <author>version_author</author>
      <comments>internal_info_about_function</comments>
   </version>
   <description>
      description_of_what_function_does--appears_in_function_catalog
   </description>
   [<imports>
         <import library="string" alias="string"/>...
     </imports>]
   <parameters>
```

```
    [<fixed-parameter name="string" type="string"/>] ...
  </parameters>
  (EPL in a code element goes here)</function>
```

The top level `function` element must specify the following three attributes:

- `name` — Logical name of the function. To avoid function conflicts in Event Modeler, the value of this attribute must be unique across all `.fdf` files in each directory.

  When you write the EPL code that implements the function, you specify `#name#` in place of the name of the function. When you use the function in a scenario, the Event Modeler replaces `#name#` with the value you specify for the `function name` attribute. When the Event Modeler does this, it adds an identifier to the name you specify to ensure that the function name is unique.

- `display-string` — Function name that the rules editor displays. When you want to use this function in a rule, this is the name that you select from the menu of functions. You might want to give your function a short name, but specify a more descriptive name for the value of the `display-string` attribute.

- `return-type` — Type of the value returned by the function. The table below shows the values you can specify for the `return-type` attribute, and the Event Modeler types these values map to:

Value of return-type Attribute	Maps to This Event Modeler Type
`String`	`text`
`float`	`number`
`enumeration`	`choice`
`boolean`	`true/false`

Defining the version element

The `version` element must contain one of each of the following elements in the following order. Use the `version` element to maintain updates to your function. In the Function Catalogs panel, when you click a function, the values you specified in the `version` element (except for the contents of `comments`) appear in the middle pane.

- `id` — Identifier for this version of your function. Typically, a version number.
- `date` — Date the function was written.
- `author` — Name of the person who wrote the function.
- `comments` — Any information about the function that you want to provide. This information appears only in the `.fdf` file; it does not appear in the Event Modeler.

For example:

```
<version>
   <id>1.0</id>
   <date>7 November 2006</date>
   <author>Matthew Amos</author>
   <comments>External function</comments>
</version>
```

Defining the description element

After the `version` element, there is a `description` element that describes what the function does. The text you enter in the `description` element appears in the middle pane of the Function Catalogs panel. For example:

```
<description>
   Convert a string to a number, and return the number.
</description>
```

Defining the imports element

The optional `imports` element provides a place to specify any plug-ins required by your function. Any plug-ins you specify must be written in the correlator plug-in API. The `imports` element can contain any number of `import` elements. Each `import` element must contain the following attributes:

- `library` — Name of the file that contains the plug-in required by your function.
- `alias` — Name of the plug-in in the `code` element of the function definition file. When you write the EPL code that implements the function, you specify `#`*alias_value*`#` as the name of the plug-in. When you use the function in a scenario, the Event Modeler replaces `#`*alias_value*`#` with the name of the function in the specified library.

For example:

```
<imports>
   <import library="TimeFormatPlugin" alias="timePlugin"</import>
</imports>
```

In the `code` element, you would specify something like the following:

```
return #timePlugin#.formatTime
```

Defining the parameters element

After the `description` element, or `imports` element if there is one, there is a `parameters` element. The `parameters` element defines the function's parameters. A function can have

- No parameters. The `.fdf` file must still contain the `parameters` element, but it is empty. For example:

  ```
  <parameters/>
  ```

- A sequence of one or more fixed parameters. Each fixed parameter has a specified name and a specified type. In the function code, you must specify any fixed parameters in the same order in which you define them in the `parameters` element.

To define fixed parameters, specify one or more `fixed-parameter` elements. Each `fixed-parameter` element contains a `name` attribute and a `type` attribute. The value of the `name` attribute indicates the name of the fixed parameter. The value of the `type` attribute indicates the type of the fixed parameter and must be `string`, `float`, `enumeration`, or `boolean`. For example:

```
<parameters>
   <fixed-parameter name="condition" type="boolean" />
   <fixed-parameter name="true_result" type="string" />
   <fixed-parameter name="false_result" type="string" />
</parameters>
```

When you display functions in the Event Modeler `Catalogs` panel, you can click on a function and then expand parameters to view the parameters required by that function. When you execute the function, each fixed parameter is required.

Defining EPL code in function definition files

In a function definition file, the last element in the `function` element is the `code` element. The `code` element contains one `CDATA` section that contains EPL code that defines one action. The requirements for the EPL code are as follows:

- The parameters and types that the EPL defines must match the parameters and types specified in the `parameters` element.
- The return type specified in the EPL code must match the type specified for the `functionreturn-type` attribute.
- Specify the name of the action as `#name#`.
- Specify the name of a plug-in as `#`*alias_value*`#`.
- The function must be valid EPL code.

  For example:

```
<code><![CDATA[
  action #name#(float f) returns float {
    return f.abs();
  } ] ] >
</code>
```

- The function can use local variables. To use a scenario variable, assign its value to a function parameter.

File Definition Formats

Block definition file format

This section describes the format of the block definition file (`.bdf`). This is a readable XML text document. Block definition files are generated automatically by Apama Studio. When these files are generated, Apama Studio creates all the XML code for specifying the block's metadata and defining its interface. The task of the developer is to add the code that implements the block's behavior.

All editing of `.bdf` files should be done in the Apama Studio block editor.

File Definition Formats

Block definition file DTD

The document must comply with the XML Document Type Definition `bdf.dtd`. This file is included in the Apama installation's `etc` directory. This description of the file format is presented for troubleshooting purposes and general background information.

When you create a new block as part of a project in Apama Studio, the best practice is to locate it in the project's default blocks directory. This directory is found in the project's `catalogs` directory. The block directory has a name in the form <*project_name*> `blocks`. So, for example, the default block directory of a project named `My_Project` will be `catalogs\My_Project blocks`.

If you place your block in the Apama Studio project's default block directory, scenarios created in the project will automatically find them and make them available in Event Modeler when you are displaying the scenario.

Apama Studio assigns the name of the file as follows:

Block Name **v** *version_number*.**bdf**

For example, the block whose `<name>` attribute is `Database Retrieval` would be defined in the file `Database Retrieval v1.0.bdf` and stored in a folder called `Database Retrieval.bdf`. This convention makes it easy to browse multiple versions of the block within a block catalog when using the Event Modeler. Note that this naming and folder placement (and creation) is all done automatically by Apama Studio.

Block definition file encodings

Apama Studio and Event Modeler always read and write block definition files in UTF-8.

XML elements that define a block

Here are the list of XML element needed to define a block, arranged to show the hierarchical ordering. The elements are described in the table that follows the list:

```
<block>
  <version>
    <id> </id>
    <date> </date>
    <author> </author>
    <comments> </comments>
  </version>
  <description> </description>
  <properties parallel-aware="false" deprecated="false">
    <input-feeds>
      <feed>
        <description> </description>
        <field>
          <description> </description>
          <validation> </validation>
        </field>
      </feed>
    </input-feeds>
    <output-feeds>
definition identical to fields in input feeds
    </output-feeds>
    <parameters>
      <field>
definition identical to fields in input and output feeds
      </field>
    </parameters>
    <operations>
        <operation>
          <description> </description>
        </operation>
    </operations>
  </properties>
  <code> </code>
</block>
```

The following table lists and describes the XML elements used to define a block:

Element	Description
<block>	The root element in any .bdf file. This element has a single text (CDATA) attribute, <name>, which must define the name of the block. This element must contain the <version>, <description>, <properties>, and <code> child elements.
<version>	The block's version. This element must contain the <id>, <date>, <author>, and <comments> child elements.
<id>	From an XML point of view, this element can contain any character data (#PCDATA), but it should be set to indicate the version number of the block, for example, 1.0 or 1.1. The version number is used to distinguish different versions of the block in the catalog browser within the Event Modeler. This version number must be the same as that encoded within the .bdf filename itself. For this reason, if the block is generated by the Block Builder, the content of this element is automatically used to name the .bdf file, in conjunction with the <name> element; see the description of the <block> element. This element has no attributes.
<date>	The date when the block was authored. This information is just for the block author's future reference. This element takes any character data (#PCDATA). It has no attributes.
<author>	The block's author. This information is just for future reference. This element takes any character data (#PCDATA). It has no attributes.
<comments>	Describes any changes that have been made to the block in this version. This element takes any character data (#PCDATA). It has no attributes
<description> — child of <block>	Can contain any character data (#PCDATA) that informatively describes the purpose of this block. As this information is displayed within the block catalog browser in the Event Modeler, it is useful to provide a brief summary of the block's functionality. It has no attributes.
<properties>	Describes the interface of the block. This element must contain the <input-feeds>, <output-feeds>, <parameters>, and <operations> child elements. This element can also contain the two Boolean attributes "parallel-aware" and "deprecated". When the parallel-aware attribute is set to true, the block can be used in a parallel scenario. When the deprecated attribute is set to true, the block has been deprecated.
<input-feeds>	List all the input feeds of this block. This element can include zero or more <feed> child elements within it. It has no attributes and cannot contain any text.
<feed>	Represents either an input feed or an output feed, depending on where it occurs within the XML document. <feed> has two attributes, id and name. id is optional. If supplied, it must be a unique string that distinguishes the

Element	Description
	feed from all other input or output feeds. The `name` attribute must also be unique, but only across input feeds or output feeds. The block definition in the EPL code defines an `action` type definition that corresponds to this feed and that takes an argument for each field in the feed. This element must contain the `<description>` and `<field>` child elements.
`<description>` — child of `<feed>`	Describes the purpose and use of the feed and is displayed by the block catalog browser in the Event Modeler.
`<field>`	The `<feed>` element can include any number of `<field>` elements. Each represents a field within the feed in question. The action in the corresponding EPL code that updates according to an input feed or sends data to an output feed must accept an argument for each field in the feed. The arguments must be in the same order as the fields defined in the XML document. A `<field>` element has two attributes, `id` and `name`. It is highly recommended to include the `id` attribute, it is optional only for backwards compatibility. It must be a unique string that distinguishes the field from all other input or output fields. `name`, a string, must also be unique but only within the feed the field belongs to. This element must contain the `<description>` and `<validation>` child elements.
`<description>` — child of `<field>`	Describes the purpose and use of the field and is displayed by the block catalog browser in the Event Modeler.
`<validation>`	Although the DTD indicates this element is optional, this is just for backwards compatibility with older blocks. This element is required, and will be added automatically with default values applied when the block is used in the Event Modeler if a `<validation>` is unspecified. This element defines the type of the field. If the field is of the scenario type string, float, integer or boolean, then no child elements are required within the `<validation>` element, whereas if the field is of type enumeration, then an `<enumeration>` child element should be included. Note that the first four types correspond to the types of the same name in the EPL code, whereas enumeration is really a string in the EPL code. `<validation>` includes nine attributes, whose relevance depends on the value entered for the first attribute, `type`. This can only take the values `string`, `float`, `integer`, `enumeration` or `boolean`, and is required. The other attributes, which are all optional, are `minlength`, `maxlength`, `minvalue`, `maxvalue`, `unique`, `mutability`, `stringcase`, and `trim`. Note that these constraints are not enforced in this version of Event Modeler and are therefore not documented.

Element	Description
<output-feeds>	Lists all the output feeds of this block. To do this, you can include zero or more <feed> child elements within it, in the same way as for <input-feeds>.This element has no attributes and cannot contain any text.
<parameters>	This element should list all the configuration parameters of the block. The functionality of a block should be configured primarily through parameters. Like the fields in input and output feeds, the whole set of parameters must correspond to an initialization event whose field parameters correspond to the block parameters, in the same order. Furthermore, for each parameter there must be an event which enables that parameter to be set independently of the others and after the initial configuration. This element takes no attributes and contains zero or more <field> child elements, one for each block parameter.
<field> — child of <parameter>	Each <field> child element corresponds to an actual parameter of the block, and the XML definition is identical as that for fields in input or output feeds. As described elsewhere, each <field> further embeds a <validation> element, where the <type> attribute is the most relevant. The type used here must correspond to the equivalent type in the EPL code.
<operations>	Represents any operations implemented in the block. Operations are chunks of functionality written in EPL that could be invoked by a scenario. This element has no attributes, and contains zero or more <operation> child elements.
<operation>	Describes an operation defined in the block. There should be an instance of this element for each operation in the block. This element takes two attributes, id and name. Both attributes are XML CDATA elements. id is optional only for backwards compatibility reasons, and should be specified. If not supplied, id will automatically be added in a way that makes the operation element unique. name, the string name for the operation, should also be made unique across the set of operations. In addition, each <operation> element should contain a <description> child element. This element can contain any character data that constitutes a relevant description of the functionality that is being made available. Its description is displayed by the block catalog browser in the Event Modeler. Note that the XML definition of an operation consists solely of a name and a description. If you wish to pass parameters to an operation, you should use the block parameter mechanism.
<code>	The actual EPL template code that implements the interface and functionality of the block. For XML validation purposes, any character data can be supplied here (#CDATA), although the content must in fact be very carefully written. The contents of this section, which can only partly be generated by Apama Studio, are discussed in detail in "Creating Blocks" on page 81.