Introduction to Apama

5.1.0

November 2013

This document applies to Apama 5.1.0 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.

Copyright © 2013 Software AG, Darmstadt, Germany and/or Software AG USA Inc., Reston, VA, USA, and/or Terracotta Inc., San Francisco, CA, USA, and/or Software AG (Canada) Inc., Cambridge, Ontario, Canada, and/or, Software AG (UK) Ltd., Derby, United Kingdom, and/or Software A.G. (Israel) Ltd., Or-Yehuda, Israel and/or their licensors.

The name Software AG and all Software AG product names are either trademarks or registered trademarks of Software AG and/or Software AG USA Inc. and/or its Subsidiaries and/or its Affiliates and/or their licensors. Other company and product names mentioned herein may be trademarks of their respective owners.

Detailed information on trademarks and patents owned by Software AG and/or its subsidiaries is located at http://documentation.softwareag.com/legal/.

Use of this software is subject to adherence to Software AG's licensing conditions and terms. These terms are located at http://documentation.softwareag.com/legal/ and/or in the root installation directory of the licensed product(s).

This software may include portions of third-party products. For third-party copyright notices and license terms, please refer to "License Texts, Copyright Notices and Disclaimers of Third Party Products." This document is located at
http://documentation.softwareag.com/legal/ and/or in the root installation directory of the licensed product(s).

Document ID: -apama-intro-5.1-20131115

Table of Contents

Preface

About this documentation

Introduction to Apama® is for new Apama users. It provides an overview of Apama, describes Apama architecture, discusses Apama concepts and introduces Apama Studio, which is the main Apama development tool. In addition to reading the material here, you can do the following to become familiar with Apama:

- Work through Apama tutorials in Apama Studio. Click Tutorials on the Apama Studio Welcome page. (Select Help > Welcome to display the Welcome page.)
- Look at Apama sample applications in Apama Studio. Click Samples on the Apama Studio Welcome page.
- Use the skills you learned in the tutorials to try modifying the sample applications as suggested in their `readme` files.

Preface

How this book is organized

The information in this book is organized as follows:

- "Apama Overview" on page 11
- "Apama Architecture" on page 19
- "Apama Concepts" on page 35
- "Getting Ready to Develop Apama Applications" on page 43

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 5.1 > 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 Java 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 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 Java in-process API 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:

Title	Contents
	• 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 JMS messaging to reliably send and receive JMS messages in Apama applications.
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

1 Apama Overview

To become familiar with Apama®, the recommended path is to work through the Apama Studio tutorials and then read this document, *Introduction to Apama*, which provides a high-level overview of Apama architecture, concepts, and development.

To view the tutorials, open Apama Studio and select Tutorials from the Welcome page. This displays links to interactive tutorials that provide step-by-step instructions for writing simple Apama applications that you can then run and monitor. Three tracks of tutorials represent the three ways that you can develop Apama applications:

- Event Modeler — Apama's GUI for creating event processing applications is for non-programmers or for programmers who want to do rapid application development.
- EPL — Apama's Event Processing Language (EPL), which is the new name for MonitorScript, is designed for developing event processing applications. This approach is for programmers who need a powerful event processing language.

  Within the product, both EPL and MonitorScript are used and should be treated as synonymous.
- Apama Java in-process API — Apama's Java interface lets programmers use the industry standard Java programming language to develop event processing applications.

Depending on what you are trying to accomplish, you can use one, two, or all three approaches in a single Apama application.

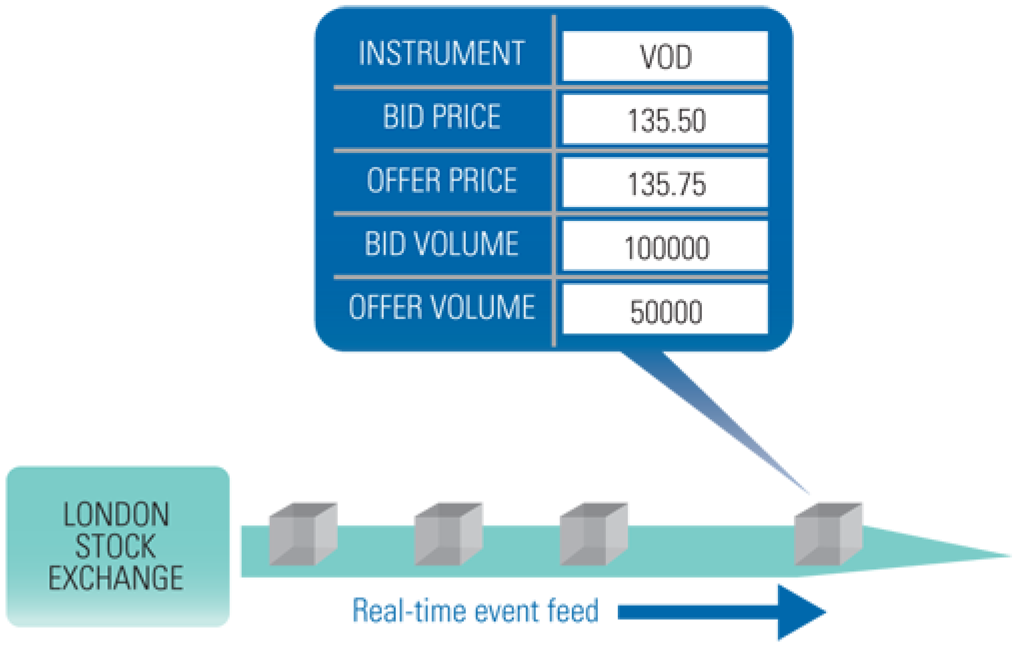
In addition to the tutorials, you can view and modify demonstration applications in Apama Studio. From the Apama Studio Welcome page, select Samples.

What is Apama?

Apama is an event processing platform. It monitors rapidly moving event streams, detects and analyzes important events and patterns of events, and immediately acts on events of interest according to your specifications.

Event-based applications differ from traditional applications in that rather than continuously executing a sequence of instructions, they listen for and respond to relevant events. Events describe changes to particular real-world or computer-based objects, for example a new bid price for Vodafone's stock on the London Stock Exchange.

Events are collections of attribute-value pairs that describe a change in an object. For example, the figure below shows stock quote events. Each stock quote has a number of attributes, including current bid price, current offer price, and current volumes. In the figure, the highlighted event shows the latest quote for Vodafone stock.



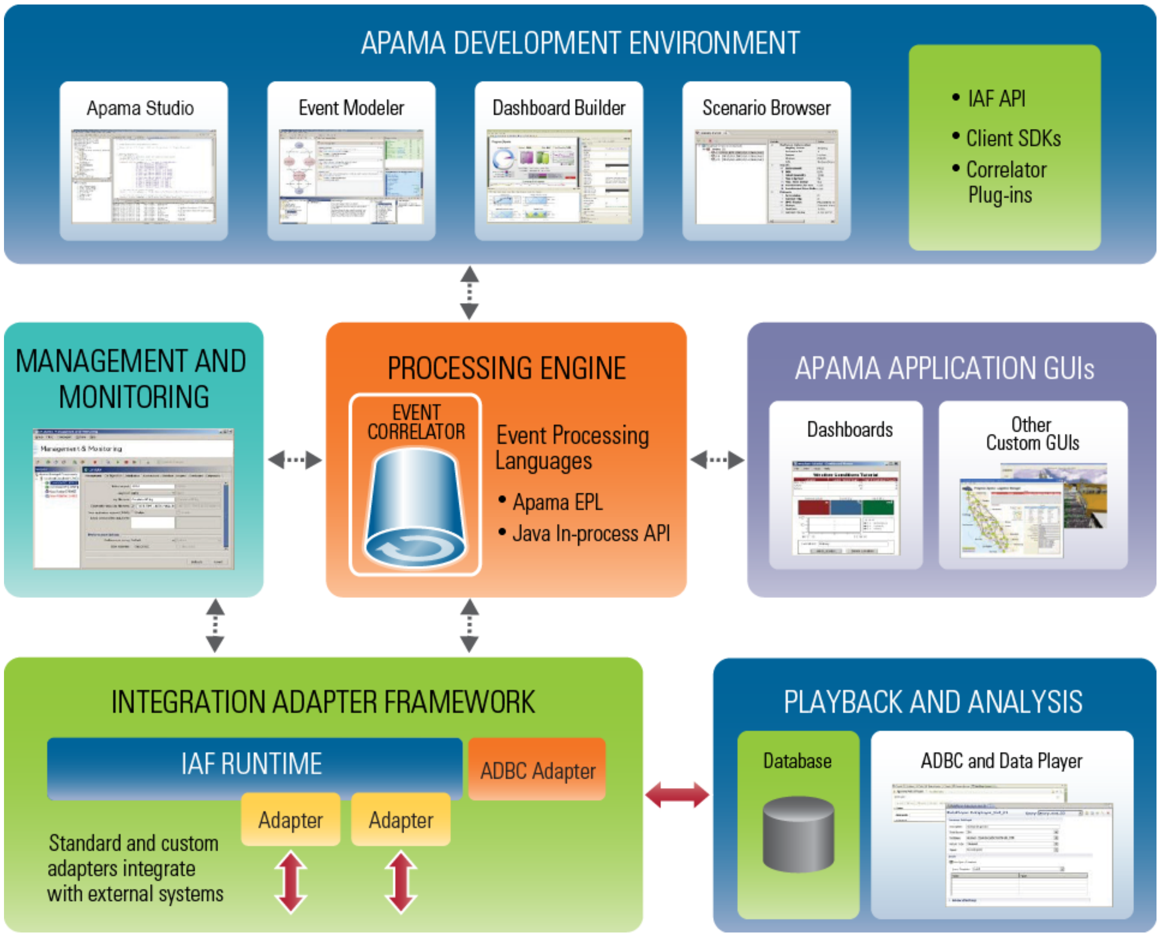
The attributes, or fields, of an individual event class may be of a variety of data types, including numerical and textual data. Events with multiple fields can be viewed as multi- dimensional data types, in that a query to find an event of interest might involve querying across several of the event fields.

Rather than executing a sequence of activities at some defined point, an event-based system waits and responds appropriately to an asynchronous signal as soon as it happens. In this way, the response is as immediate (or real-time) as possible.

The main Apama features include:

- Graphical development tools accessible to business users.
- EPL, which is a concise, richly-featured event processing language.
- Integration Adapter Framework (IAF), which provides easy integration to external event source and systems.
- Sophisticated analytics with native support for temporal arguments.
- Sub-second response to detected events and patterns of interest.
- Highly scalable, patented, event-driven architecture, supporting tens of thousands of concurrent scenarios.
- Integrated tools for creating visually appealing user dashboards.
- Flexible event replay for testing new event scenarios and analyzing existing ones.
- Tools for managing and monitoring your application.

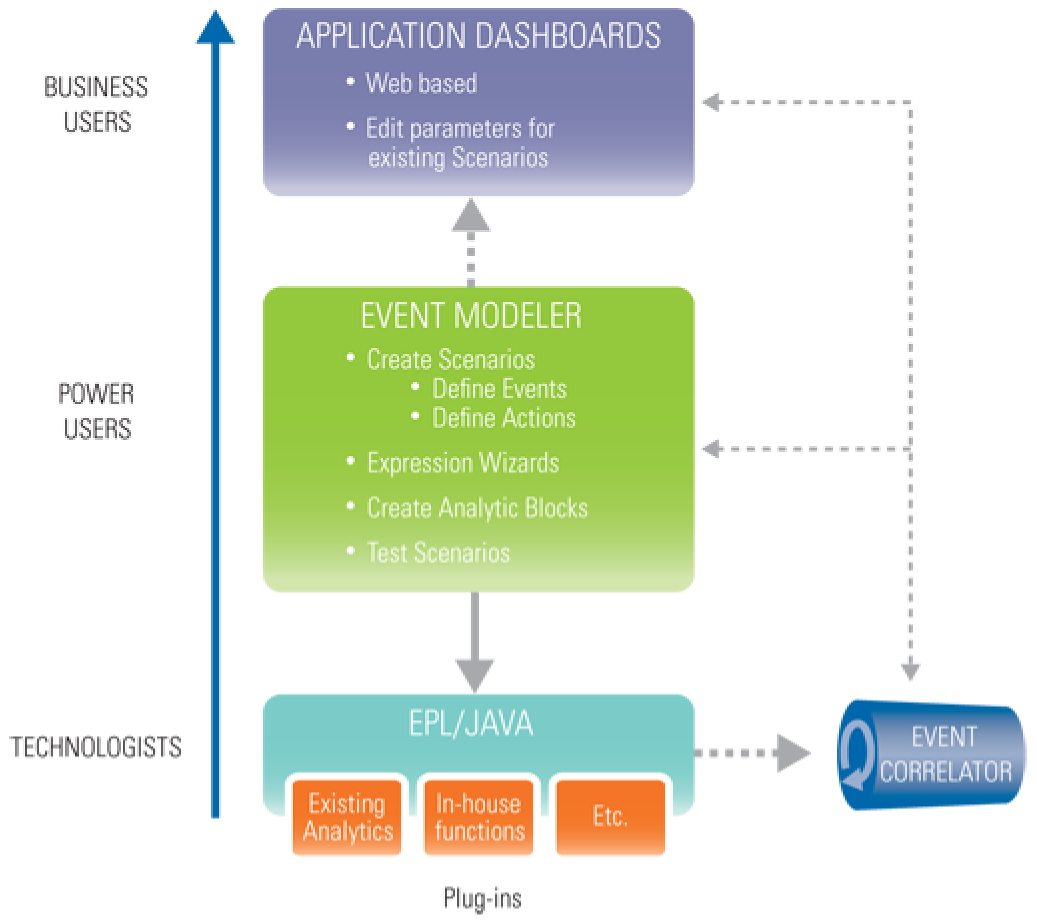
The following functional diagram shows the main Apama features:



Apama Overview

Understanding the different user viewpoints

Apama has been designed for a range of users. The figure below shows the spectrum of users from technologists, business technology users, referred to as power users, up to pure business users. Apama provides different facilities for each of these classes of user. After the initial design is set for an Apama application, multiple users can work concurrently to implement the design.



Technologists can make use of the full set of APIs and technologies within the Apama architecture to create sophisticated, custom, CEP solutions. Using Apama Studio they can create applications directly in EPL or Java. They can extend the capabilities of the Apama correlator with their own in-house analytic routines. Using the Integration Adapter Framework (IAF) they can integrate with a new data service by developing a new adapter if one that can be plugged in does not already exist. They can also take advantage of low-level APIs for building custom client user interfaces in C, C++, Java and .Net.

Power users are provided with GUI tools (Event Modeler and Dashboard Builder) to enable the creation of scenarios without having to write code. A scenario is a sequence of event monitoring and event responses that constitutes a single business activity. A scenario can be an application in its own right, or part of a bigger application. Using the Event Modeler graphical tool, you can create scenarios from reusable blocks.

Thus, there are three approaches to developing Apama applications. Your development team can use one, two, or all three in an Apama application:

- EPL — Apama's native event processing language
- Java — Apama provides a Java in-process API for processing events
- Event Modeler — Apama's GUI for creating event processing scenarios

Pure business users are often only interested in the end-game application. The output of the Dashboard Builder tool provides an immediately usable application for this purpose.

In addition, Apama provides management and monitoring tools for users whose primary responsibility is to manage deployed Apama applications.

Apama Overview

About Apama license files

Your Apama sales contact or an authorized Apama reseller must supply you with an Apama license file. There are instructions in the installation guide for copying the license file to the correct location.

It is possible to run Apama for a limited time without a license file or with an expired license. Apama behavior with regard to the Apama license file is as follows:

- You can start a correlator without specifying the `-l` *`license.txt`* option. When you start the correlator without specifying a license file, the correlator runs for 30 minutes in a constrained mode that accepts connections from only the local host (`127.0.0.1`).
- A running correlator does not shut down when its license expires. It continues operation for seven days beyond expiration. The correlator logs periodic warning messages until it reaches the end of the seven days or until you replace the expired license.
- Removing the license file from a running correlator does not cause it to shut down. It continues operation for seven days after the license file is removed. The correlator logs periodic warning messages until it reaches the end of the seven days or until you restore the license.
- You can start a correlator with an expired license if it is less than seven days beyond expiration.
- You can use the Management and Monitoring console to start a correlator without first sending a license, but only on the same host as the console. Also, you must configure the component's host as `localhost`.

Refer to the licencing terms specified in your software contract for any additional legal restrictions that may be imposed on your use of Apama.

If you obtain a license after you have been running Apama, copy the `license.txt` file to the `license` directory in your `APAMA_WORK` directory, which is typically in your home directory, for example: `C:\Users\tcohen\SoftwareAG\ApamaWork_5.1\license\license.txt`.

Apama Overview

Software AG Knowledge Center

Use Software AG's Empower customer support portal to search the Knowledge Center, manage your Knowledge Center subscriptions, download products, view product version availability information, product documentation and more.

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

If you have any questions, please 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.

Apama Overview

2 Apama Architecture

Apama architecture has a modular, scalable design with core features that

- Monitor inbound events typically delivered by a messaging infrastructure or market data feed.
- Analyze those events in memory, either singly or in conjunction with other events whose attributes and temporal ordering represent a pattern.
- Trigger outbound events that represent an action to be taken in response to the analysis.

As you can see, Apama's architecture is designed to process events. Event processing requires an architecture that is fundamentally different from traditional data processing architectures. Because Apama's architecture is event driven, an understanding of the distinctive qualities of this architecture is crucial to designing and building robust Apama applications.

Distinguishing architectural features

Apama inverts the paradigm of traditional data-centric systems. Rather than the store > index > query model of those architectures, Apama introduces the correlator — a real-time, event processing engine. When you develop an Apama application, you create monitors that specify the events or patterns of events that interest you. These specifications are the logical equivalent of database queries. After you load monitors into the correlator, incoming events flow over them and they monitor incoming event streams for the events and patterns you specified. When a monitor finds a matching event or pattern, it processes it according to the rules you specify in the monitor.

Apama's architecture is further distinguished by its ability to support huge numbers of individual monitors operating simultaneously. Each can have its own logic for monitoring the event streams, seeking out patterns and, upon detection, triggering specified actions.

The correlator supports two main programming languages: EPL and Java. EPL, Apama's native event programming language, lets developers define rules for processing complex events. Such rules let the correlator find temporal and causal relationships among events. Developers who prefer to program in Java can use the correlator's Java API. In addition, the Event Modeler GUI lets non-programmers define scenarios for handling events. Event Modeler translates scenarios into monitors. EPL, Java, and Event Modeler are three different approaches for developing Apama applications. A development team can use one or more of these.

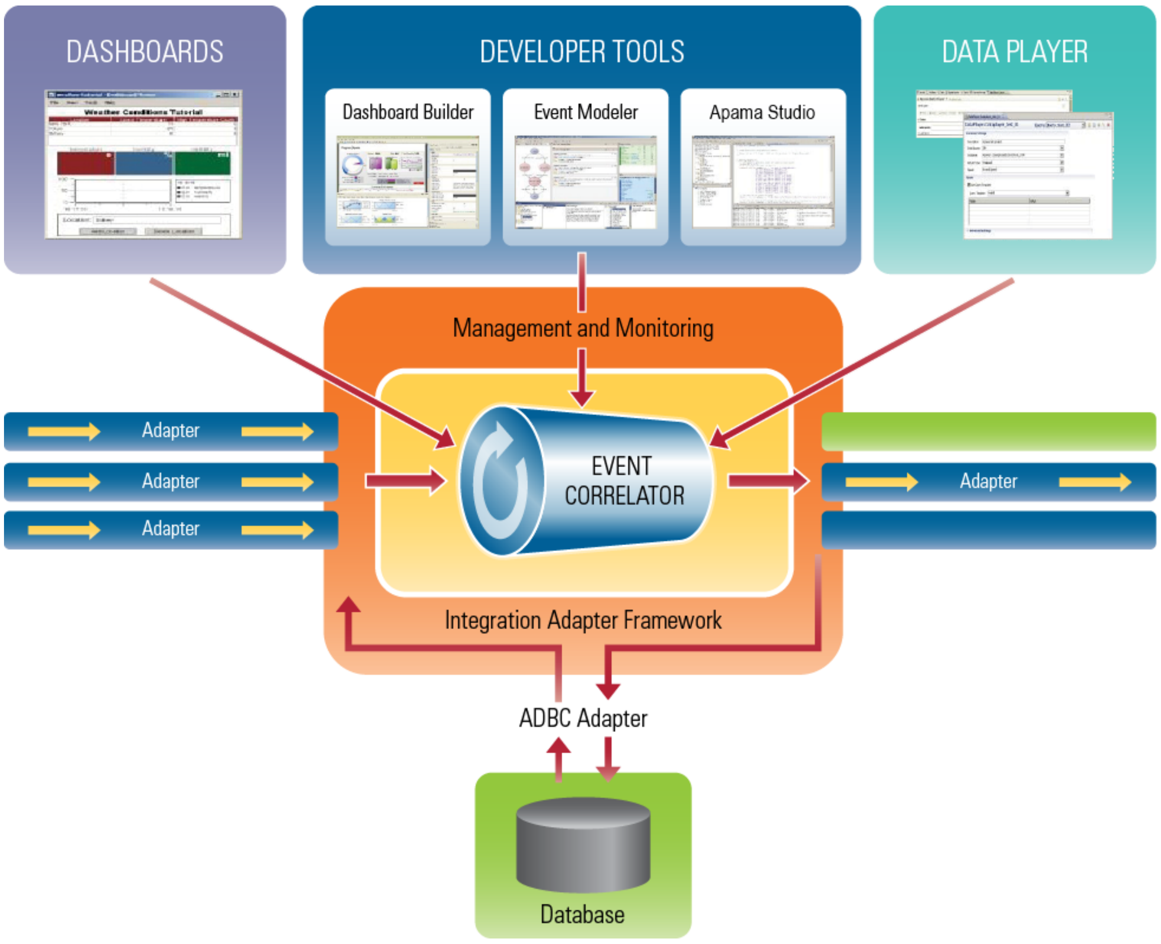
Messages on a variety of transports, such as an Enterprise Service Bus (ESB), carry events to and from event correlators. Apama adapters translate application-specific data into Apama application event formats that the correlator can process. For example, Apama trading systems integrate with various exchanges by means of adapters that translate between Apama and market data feeds or order management protocols.

Apama's architecture also provides tools for creating dashboards that let you manage your event processing scenarios. You can use Apama dashboards to start, stop, parameterize, and monitor event processing from client applications.

The Apama ADBC (Apama Database Connector) adapters provides a mechanism to capture and to replay event streams from JDBC/ODBC compliant third party databases. Together, the ADBC standard adapters and Apama Studio's Data Player let you analyze the actual performance of scenarios already in production, and also investigate the likely behavior of Apama scenarios prior to deployment.

Finally, Apama's architecture provides a monitoring and management service that enables centralized configuration of an Apama deployment. A monitoring and management console lets you configure and assess the status of all Apama components in your deployed system.

The following figure illustrates Apama architecture. Each component is described later in this section.



Apama Architecture

How Apama integrates with external data sources

You can connect Apama to any event data source, database, messaging infrastructure, or application. There are three ways to do this:

- Implement Apama Integration Adapter Framework (IAF) adapters.
- Develop client applications with Apama Java, .NET, and C++ APIs.
- Create applications that use the correlator-integrated Java Message Service (JMS) connectivity.

Using IAF adapters to connect with external data sources

Apama's Integration Adapter Framework (IAF) provides bi-directional connectivity with event sources and with your environment.

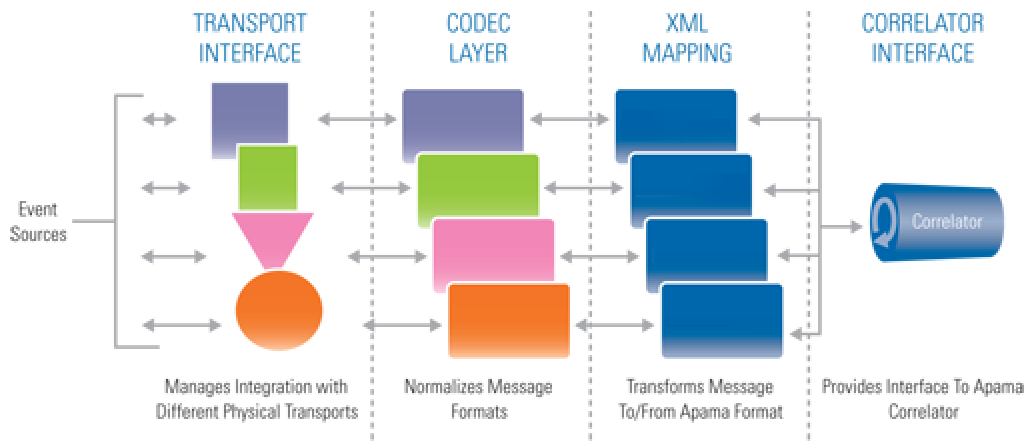
Apama adapters provide both connectivity and XML-based mapping between your application's data format and Apama's internal format. The purpose of an adapter is to translate events from a proprietary format into Apama events. This lets the correlator analyze those events. Also, an adapter converts Apama events into your proprietary source format. This lets Apama send the events to an external service.

Adapters allow a single Apama application to efficiently monitor and analyze disparate event types within a common event processing scenario. For example, the same scenario can process events relating to foreign exchange (FX) aggregation, smart order routing and cross-asset trading in capital markets or cold chain automation in supply chain applications.

Within the IAF, Apama offers a range of standard adapters for capital markets, infrastructure, connectivity to data and messaging sources, and APIs for building custom adapters.

The IAF is a server component that adapters plug into for run-time invocation. You can develop an adapter with the IAF adapter library, along with whatever specific connection APIs you need to connect to your data service. Each adapter is structured so that the mapping of parameters between the source format and Apama format can be configured dynamically through XML.

The following figure shows the bi-directional operation of an adapter.


TRANSPORT INTERFACE
CODEC LAYER
XML MAPPING
CORRELATOR INTERFACE
Event Sources
Correlator
Manages Integration with Different Physical Transports
Normalizes Message Formats
Transforms Message To/From Apama Format
Provides Interface To Apama Correlator

Examples of types of adapters include:

- Middleware messaging adapters

  Several middleware bus technologies are available on the market, including technologies by Tibco, IBM MQ Series, Vitria, WebMethods, SeeBeyond and others. When the middleware you are using is JMS-compliant, you can create applications that use the correlator-integrated JMS in place of an adapter.

  Apama is able to interface with these technologies through an appropriate adapter. If the middleware bus offers publish and subscribe capabilities, then Apama can become a named endpoint like any other service. Apama is able to receive events from the bus and convert them, via an adapter, into Apama events for the correlator to process. An adapter can convert any events emitted by a correlator back into the native bus format.

- Database adapters

  Apama is able to connect to databases for a number of purposes, including querying historical state or storing key events as an audit trail in the corporate database. Most popular databases support a standard access protocol, such as ODBC or JDBC. The Apama Database Connector

(ADBC) provides ODBC and JDBC adapters that use these standard access protocols to connect to your database.

- Web Services adapter

  The Apama Web Services Client adapter is usable across all Apama components. Interaction with the IAF is transparent to the application.

- Custom real-time feed adapters

  A number of companies provide real-time content as information feeds. Examples in the finance industry include Reuters, who provide a variety of stock and news feeds, and GLTrade, who provide bi-directional access to a variety of the world's equities and derivatives exchanges. Many such companies use custom communications protocols to provide their data. However, Apama adapters have been easily developed for these and other bi-directional data services.

For complete information, see "The Integration Adapter Framework" in *Developing Adapters*.

Using Apama APIs to connect with external data sources

A range of APIs let you extend Apama at the dashboard, client, and correlator levels for integration with other environments, such as Java, .NET, C, or C++. In addition, you can extend correlator behavior with C and C++ plug-ins that can call external function libraries from within an application.

Using correlator-integrated JMS to connect with external data sources

Apama's correlator-integrated JMS messaging provides an efficient way to receive and send JMS messages to and from Apama applications. It also provides for reliable messaging (guaranteed delivery) and duplicate detection.

Apama Architecture

Descriptions of Apama components

While traditional architectures can respond to events after they have happened, Apama's event-driven architecture responds in real time to fast moving events of any kind. Apama applications leverage a platform that combines analytic sophistication, flexibility, performance and interoperability. In addition to being an event processing engine, Apama provides sophisticated development tools, a flexible testing environment, an extensible integration framework and graphically-rich dashboards. This makes Apama a comprehensive event processing platform for building real-time, event-driven applications.

The following topics describe the main Apama components:

- "Description of the Apama correlator" on page 24
- "Description of event processing languages" on page 24
- "Description of Apama Studio" on page 27
- "Description of Event Modeler" on page 27
- "Description of Dashboard Builder and Dashboard Viewer" on page 28
- "Description of client development kits" on page 28
- "Description of Management and Monitoring Console" on page 29

- "Description of Apama Studio's Data Player" on page 30

For information about the Integration Adapter Framework, see "How Apama integrates with external data sources" on page 21.

Apama Architecture

Description of the Apama correlator

Apama's event correlator is the engine that powers an Apama application. Correlators execute the sophisticated event pattern-matching logic that you define in your Apama application. Apama applications track inbound event streams and listen for events whose patterns match defined conditions. The correlator's patented architecture can monitor huge volumes of events per second

When an event or an event sequence matches an active event expression, the correlator executes the appropriate actions, as defined by the application logic.

The correlator

- Can concurrently search for and identify vast numbers of discrete event patterns with sub-millisecond responsiveness.
- Can deliver low latency analytics on multiple inbound data streams by monitoring the event streams for patterns you specify.
- Goes beyond simple event processing to deliver actionable responses.

See also "How the correlator works" on page 30.

Descriptions of Apama components

Description of event processing languages

Apama provides developers with two language models for building event-based applications:

- EPL, which is Apama's native event processing language
- Apama (in-process) Java API

This section gives you a flavor for how these languages process events. You can find complete information here:

- *Developing Apama Applications in EPL*
- *Apama EPL Reference*
- *Developing Apama Applications in Java*
- *Apama Java API Reference*

Descriptions of Apama components

Introduction to Apama EPL

Before EPL can look for patterns in event streams, you must define the types of events you are interested in and inject their definitions in the correlator. An event definition informs the correlator about the composition of an event type. An example event definition for a stock exchange tick feed is as follows:

```
event StockTick {
   string symbol;
   float price;
   float volume;
}
```

Each field of the event has a type and a name. The type informs the correlator how to handle that field and what operations to allow on it. As you can see, the correlator can handle multiple data types, such as numeric values and textual values, within the same event type. Apama can handle any number of different event types at one time.

Client event sources and the IAF need to be able to inject events into the correlator. For the correlator to be able to detect an event of interest, the event's type definition must have been loaded into the correlator. An example of a `StockTick` event is as follows:

```
StockTick (“APAMA”, 55.20, 250010)
```

The basic EPL structure is called a monitor. A monitor defines:

- One or more listeners —EPL provides event listeners and stream listeners.
  - An event listener observes the correlator event stream analyzing each event in turn until it finds a sequence of events that match its event expression. When this happens the event listener triggers, causing the correlator to execute the listener action.
  - A stream listener passes stream query output to procedural code. A stream query operates on one or two streams to transform their contents into a single output stream. The type of the stream query output items need not be the same as the type of the query input items. The output for one stream query can be the input for another stream query. At the end of the chain of stream queries, a stream listener coassigns each stream query output item to a variable and executes specified code.
- One or more actions — an action is one or more operations that the correlator performs. An action might be to register a listener or it might be an operation to perform when the correlator finds a match between an incoming event or sequence and a listener.

The following EPL example illustrates these concepts in the form of a simple monitor called `PriceRise`. The monitor is composed of three actions. The first two actions declare listeners, which are indicated by the `on` keyword.

```
monitor PriceRise
{
   action onload() {
      StockTick firstTick;
      on all StockTick(“IBM”,>=75.5,*):firstTick {
         furtherRise (firstTick);
      }
      float f;
      from tick in all StockTick(symbol="IBM")
         within 60.0 every 60.0
         select mean(tick.price): f { average(tick.price); }
   }
   action average(float av) {
      log "60-second average for IBM: "+av.toString();
   }
   action furtherRise(StockTick tick) {
      StockTick finalTick;
      on all StockTick("IBM",>=(tick.price*1.05),*): finalTick {
         log "IBM has hit "+finalTick.price.toString();
         emit PlaceSellOrder("IBM",finalTick.price,1000.0);
      }
   }
}
```

When a monitor starts running, the correlator executes the monitor's `onload()` action. In the `PriceRise` monitor, the `onload()` action creates an event listener for all IBM stock ticks that have a price above `75.5` at any volume and a stream listener for all IBM stock ticks. Since the last field of the event (`volume`) is irrelevant to the event listener it is represented by an asterisk (*), which indicates a wildcard. This monitor effectively goes to sleep until the correlator detects an IBM stock tick.

If the correlator detects an IBM stock tick, the stream listener takes it as input and uses it to log 60-second averages for IBM stock prices. If the IBM stock tick also has a price that is greater than or equal to 75.5, the correlator copies the field values in that event to the `firstTick` variable and calls the `furtherRise()` action.

The `furtherRise()` action creates another event listener. This event listener is looking for the next part of the event pattern, which involves detecting if the IBM stock price goes up by more than 5% from its new value. The second listener uses the `firstTick` variable to obtain the `price` value in the event that caused the first listener to detect a match. If the price rise occurs, the correlator copies the values in the matching, incoming event to the `finalTick` variable, and executes the associated block of code.

The associated block of code logs the new price and emits a `PlaceSellOrder` event to a receiver that is external to the correlator. For example, an adapter can pick up this event, and translate it into a message that an order book can operate on. The `PlaceSellOrder` event causes placement of an order for 1000 units of IBM stock.

Description of event processing languages

Introduction to Apama (in-process) Java API

EPL was designed specifically for event processing. However, some organizations and individuals prefer to use a mainstream programming language, such as Java. Consequently, Apama has made the features of the correlator available in Java.

The correlator uses its embedded Java virtual machine (JVM) to execute Java monitors. The following Java code defines the `StockTick` event type.

```
import com.apama.jmon.Event;

public class StockTick extends Event {
   public String symbol;
   public double price;
   public double volume;

   //No argument constructor
   public StockTick() {
      this("",0,0);
   }

   //Constructor
   public StockTick(String name, double price, double volume) {
      this.name = name;
      this.price = price;
      this.volume = volume;
   }
}
```

In Java, an event class definition must include the following:

- A set of public variables to hold the event's fields
- A no-arguments constructor

- A parameterized constructor

While this is not as concise as EPL, these are familiar Java conventions.

The following code defines a Java monitor that listens for any IBM stock tick events with a price that is greater than `75.5`. The `onLoad()` method creates an event expression object, which receives an Apama event string. This string represents the event to listen for. Also, the `PriceCheck` class implements the `MatchListener` class, which provides a `match()` method to be invoked if the correlator finds a match. This is passed to the event expression object as well.

```
import com.apama.jmon.*;
public class PriceCheck implements Monitor, MatchListener {
   public PriceCheck() {}
   public void onLoad() {
      EventExpression eventexpr =
         new EventExpression("StockTick(\"IBM\",>75.5,*)");
      eventexpr.addMatchListener(this);
   }
   public void match(MatchEvent event) {
      System.out.println("Pattern detected");
   }
}
```

Description of event processing languages

Description of Apama Studio

Apama Studio is the main entry point for Apama development. When you are ready to start developing your Apama application, open Apama Studio and create an Apama project to contain your application files.

See "Introduction to Apama Studio" on page 44. See *Using Apama Studio* for complete information.

Descriptions of Apama components

Description of Event Modeler

Apama Studio's Event Modeler editor provides a graphical environment that complements Apama's event processing language. You use Event Modeler to create and modify scenarios. A scenario is a description of an application that will be deployed on and leverage the real-time capabilities of Apama's event correlator. Event Modeler provides graphical tools for:

- Defining the states that the scenario passes through
- Defining rules that control what happens in each state and when there is a transition to the next state
- Defining input and output variables
- Adding standard and custom blocks that provide reusable operations

A scenario is a template that specifies how instances of the scenario will behave once they are created inside a correlator

The design philosophy behind Event Modeler is that once you define a scenario anyone should be able to create and configure instances of it without requiring technical expertise or knowledge of its inner workings. Therefore, once the scenario author has finished editing their scenario, the next step is to create a graphical dashboard for it. This lets the end users access and interact with the scenario through an intuitive and easy to manipulate graphical user interface.

See also "Understanding scenarios and blocks" on page 39.

After you define a scenario, you use Apama Studio tools to test it. Apama Studio translates your scenario into a monitor and injects the monitor into the correlator. For an introduction to monitors, see "Description of event processing languages" on page 24.

For additional information, see "Overview of using Event Modeler" in *Developing Apama Applications in Event Modeler*.

Descriptions of Apama components

Description of Dashboard Builder and Dashboard Viewer

Apama’s Dashboard Builder enables you to create end-user dashboards and prepare them for deployment. A dashboard provides the user interface to a scenario and allows you to create instances of the scenario, view the performance of the scenario and interact with it.

For applications written in EPL, you create data views and use Dashboard Builder to create a dashboard from the data views.

Dashboard Builder is a visual design environment. A primary goal of Dashboard Builder is to enable non-technical users to create sophisticated dashboards. Consequently, Dashboard Builder provides a complete design and deployment environment. With a wide range of visual objects and drag-and-drop development, Dashboard Builder provides the tools needed to create highly customized dashboards from which users can start/stop, parameterize and monitor Apama scenarios and data views.

Dashboard Builder offers an extensive array of graphical widgets with which to build custom user dashboards. Meters, gauges, tables, graphs, and scales are available for creating highly customized dashboards. You can further personalize the interface through addition/deletion of panels or modification of graphics and color schemes.

Dashboard Viewer is the tool that end-users run to access dashboards.

See also "Introduction" in *Building Dashboards* and "Concepts" in *Using Dashboard Viewer*.

Descriptions of Apama components

Description of client development kits

Apama is highly extensible with a range of APIs provided at the dashboard, client and correlator levels. You can use these APIs to integrate with other environments, such as Java, Java Beans, C, C++, or .NET. You can extend correlator behavior with plug-ins that can call external function libraries from within an application scenario.

See "The Client Software Development Kits" in *Developing Apama Clients*.

Descriptions of Apama components

Description of Management and Monitoring Console

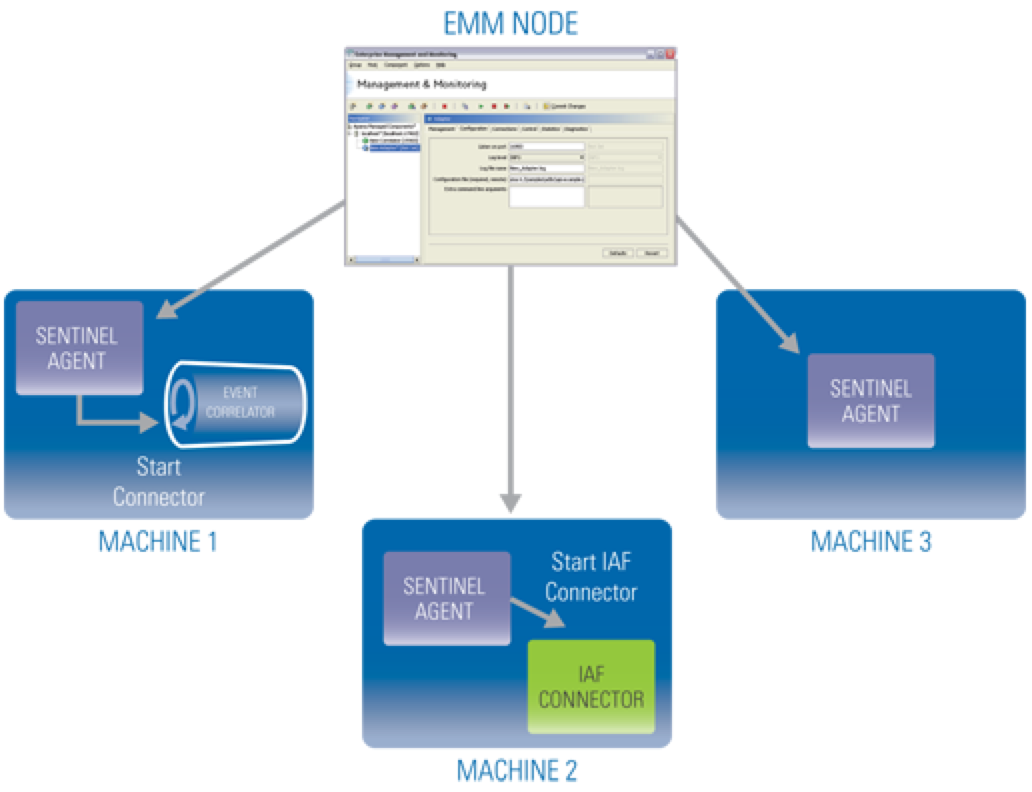
Apama's Management and Monitoring console enables coordinated management and monitoring of an entire Apama platform deployment. The console provides two basic components:

- Centralized control service, with incorporated dashboard
- Sentinel agents, running on each distributed node

The Management and Monitoring console is where you configure management policies. The Management and Monitoring component uses the sentinel agents to implement these policies on the distributed machines.

See "Using the Management and Monitoring Console" *Deploying and Managing Apama Applications*.

The following figure illustrates this. At the top of the figure, is the client machine that is running the Management and Monitoring component. This is the control node. The other three machines in the figure are running the sentinel agents. The control node uses the sentinel agents to start, shut down, recover and configure correlators, adapters, and Apama client applications.



The capabilities of the Management and Monitoring console are as follows:

- Distributed start up and configuration of components. Each sentinel controller can create components such as correlators and IAF adapters. You can use the Management and Monitoring console to configure and start a set of components, for example, three correlators.
- Component health monitoring and failure detection. Each Apama component provides an interface, through which the Management and Monitoring console can regularly ping to ensure the component is up and performing properly. Failure to receive an appropriate response can

indicate the failure of a component. This capability is configurable, since some applications require higher levels of fault tolerance than others.

- Component recovery. It is possible to set up automated policies to restart failed components. For correlators, this might involve the restoration of checkpointed state.
- Component shut down. It is possible to shut down components centrally through the Management and Monitoring console.

Descriptions of Apama components

Description of Apama Studio's Data Player

Apama Studio's Data Player accelerates the development/deployment cycle of Apama scenarios, EPL applications, or Apama Java applications by letting you pre-test (via simulation) your applications on event streams captured in Apama. It also supports flexible event processing replay features.

Data Player provides analysis tools for the Apama environment. It enables Apama users to investigate the likely behavior of Apama applications prior to deployment, as well as analyze the actual performance of those applications already in production.

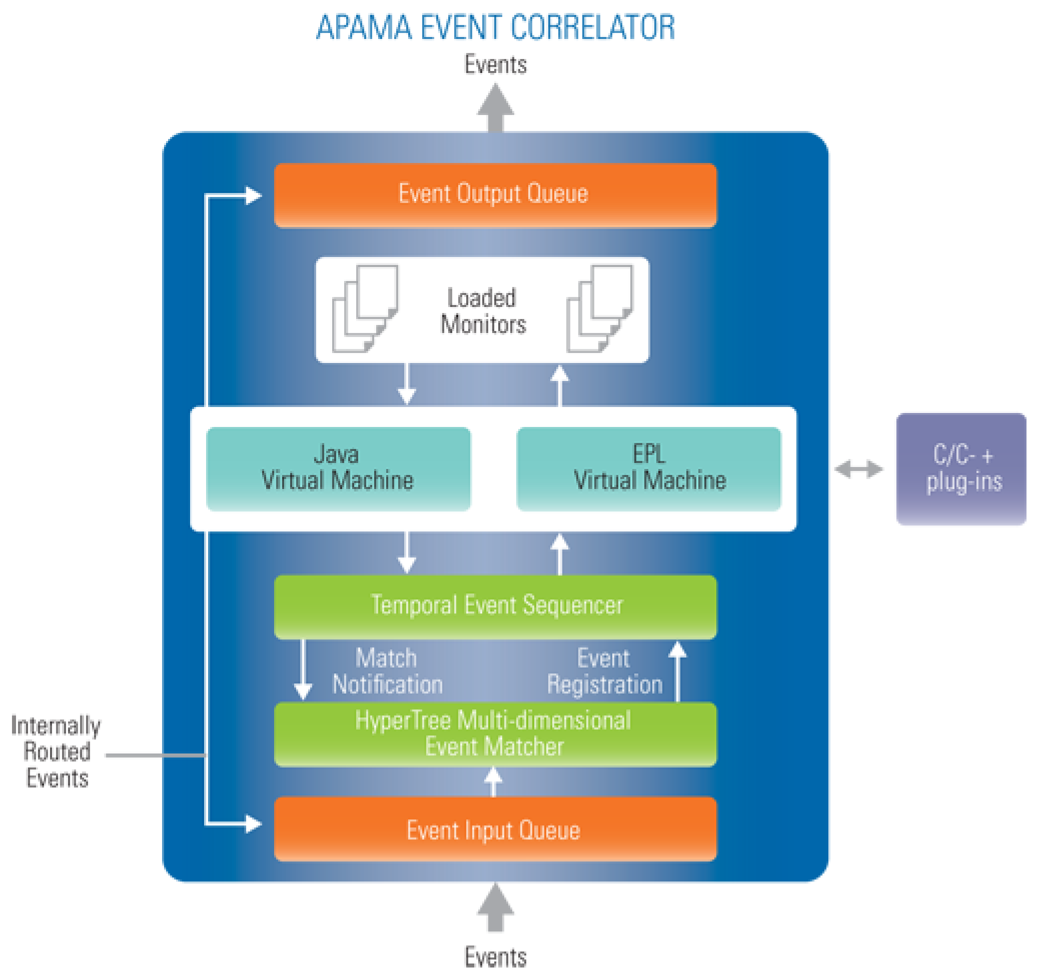
Data Player operates on data captured by the Apama Database Connector (ADBC). ADBC provides Apama standard adapters that allows access to JDBC/ODBC compliant databases as well as to Apama Sim files. Analysis can include all events received by Apama or only selected event streams. Likewise, you can choose specific segments of time from the past (for example, an entire day, a specific 30 minute period, or any user chosen time slice). Additionally, you can accelerate replay speeds many times the actual live speeds, or slow them down or pause for more careful exploration of event processing operations.

See *Using Apama Studio* for information about the Data Player. See *Deploying and Managing Apama Applications* for information about the ADBC adapter.

Descriptions of Apama components

How the correlator works

The following figure shows the inner details of a running event correlator. After the figure, there is a detailed discussion of how the event correlator works.



Monitors define events and patterns of interest and the responses to take if those patterns are detected. You can use EPL or Java to write monitors directly. When you use Event Modeler to create a scenario, Apama Studio translates the scenario into a monitor that the correlator can execute.

The correlator does not just execute loaded monitors in a sequential manner, as if they were traditional imperative programs. Instead, the correlator loads its internal components (the hypertree and the temporal sequencer) with the monitoring specifications of the monitors. The in-built virtual machines execute only the sequential analytic or action parts of the monitors.

The correlator contains the following components:

- HyperTree multi-dimensional event matcher

  The event matcher contains data structures and algorithms designed for high performance, multi-dimensional, event filtering. The correlator loads the event matcher with event templates. An event template identifies the event you are interested in. Logically, an event template is a multi-dimensional query. For example, a template for a stock market event might have values such as the following:

  - Instrument: `IBM`
  - Bid Price: `93.0 <- -> 94.5`
  - Offer Price: `*`
  - Bid Volume: `>10000`

  - Offer Volume: *

  This event template expresses a multi-dimensional search over stock market events. The template will match any event about stock IBM, which has a bid price between 93.0 and 94.5 and a bid volume greater than 100000. The offer price and volume are irrelevant to this search and so wildcards are used.

  This kind of multi-dimensional, multi-type, ranged searching is what the event matcher was specifically designed for. In checking whether an incoming event matches any of the registered event templates, the event matcher exhibits logarithmic performance. This means that vast numbers of event templates can be queried against, with the minimum possible performance tail-off.

  An event template is the basic unit of monitoring. A simple monitor might have one or a few event templates. A more complex monitor might have many. A monitor needs to load event templates only when events that match the specification are relevant to the monitor: in a multi-stage monitor, a monitor can insert and remove several event templates as the monitoring requirements change.

- Temporal and stream sequencer

  The temporal and stream sequencer builds upon the single event matching capabilities of the event matcher to provide multiple temporal event and stream correlations. With EPL or Java, you can declare a temporal sequence such as "tell me when any news article event is followed within 5 minutes by a 5% fall in the price of the stock the news article was about". This is a temporal sequence, with a temporal constraint. The sequence is a news article event, followed by the next stock price event, and then another stock price event with a price 5% less than the previous price event. The temporal constraint is that the last event occurs within 5 minutes of the first event.

  The sequencer manages this temporal monitoring process, using the event matcher to monitor for appropriate event templates. This capability saves the programmer from having to encode such complex temporal logic through less intuitive imperative logic.

- Monitors

  The correlator provides the capability for monitors to be injected as either EPL or Java bytecode. The number of monitors that can be loaded into a single correlator are only limited by memory size. When loaded, a monitor configures the hypertree and temporal sequencer with event templates for monitoring. The correlator stores the monitor internally and executes actions in the appropriate virtual machine in response to event detection.

  Each monitor instance has its own address space within the correlator for storage of variables and other state. Monitor temporary storage size is limited only by the memory size of the host machine. (Apama Studio translates Event Modeler scenarios into EPL monitors.)

- Event input queue

  External interfaces, such as adapters, inject events into the correlator. To start the monitoring process, the correlator injects each event, in the order in which it arrives, into the hypertree. Any matches filter through the temporal sequencer and invoke required actions in the virtual machines. Some actions might cause events to be queued for output. During peak event input flow, events might wait on an input queue for an extremely brief moment.

- EPL virtual machine

  In response to detected event patterns of interest, the EPL virtual machine executes EPL. The fact that the correlator behaves this way, rather than continuously executing imperative code, is another reason for its high performance. Also, you can implement parallel processing in your applications so that the correlator can concurrently execute code in multiple monitors.

- Java virtual machine

  The Java virtual machine is a standard JVM that has been embedded in the correlator. Thus any standard Java code features are accessible from monitors. The Java virtual machine behaves exactly as the EPL virtual machine in that the detection of event patterns of interest invokes code fragments.

- Event output queue

  Monitor actions can output events to be communicated to other monitors or to external systems. When a monitor routes an event, the event goes to the front of the input queue. This ensures that any monitors that are listening for that event immediately detect it. When a monitor generates an event for an external receiver the event goes to an output queue for transmission to the appropriate registered party.

  When you use the correlator in conjunction with the IAF, then an output event might represent an action on an external service. The IAF transforms the output event into an invocation of the external service. An example is an event that places an order into the order book of a Stock Exchange.

- Plug-ins

  It is possible to extend the capabilities of the correlator through a plug-in. A plug-in is an externally-linked software module that registers with the correlator through the correlator’s extension API. Plug-ins are useful when programming libraries of useful real-time functions have been built up. These functions can be made available as objects that can be invoked by EPL actions.

  Apama provides a number of standard plug-ins:

  - The MemoryStore plug-in lets monitors share in-memory data.
  - The TimeFormat plug-in helps you format dates and times.

- State persistence

  When the correlator shuts down the default behavior is that all state is lost. When you restart the correlator no state from the previous time the correlator was running is available. You can change this default behavior by using correlator persistence. Correlator persistence means that the correlator automatically periodically takes a snapshot of its current state and saves it on disk. When you shut down and restart that correlator, the correlator restores the most recent saved state.

  To enable persistence, you indicate in your EPL code which monitors you want to be persistent. Optionally, you can write actions that the correlator executes as part of the recovery process. When code is injected for a persistence application, the correlator that the code is injected into must have been started with a persistence option. Persistent monitors must be written in EPL. State in Java monitors cannot be persistent. State in chunks, with a few exceptions, also cannot be persistent.

You program the correlator by injecting monitors that you write in EPL or Java. If you use Event Modeler to define scenarios, Apama Studio translates the scenario definition files (`.sdf` extension) into monitors for you.

By default, the correlator operates in a serial manner. During serial correlator operation, the correlator processes events in the order in which they arrive. Each external event matches zero or more listeners. The correlator executes a matching event's associated listeners in a rigid order. The correlator completes the processing related to a particular event before it examines the next event. If the processing of an event generates another event that is routed to the correlator, the correlator processes all routed events before moving on to the next event in its queue. If a listener action block does not route events, the next external event is considered.

For some applications, this serial behavior might not be necessary. In this case, you can improve performance by implementing parallel processing. Parallel processing lets the correlator concurrently process code in multiple monitor instances. To implement parallel processing, you must create parallel contexts in EPL or use Event Modeler. After you create contexts with EPL, you can use them in either EPL or Java applications.

Parallel processing in the correlator is quite different from the parallel processing provided by Java, C++, and other languages. These languages allow shared state, and rely on mutexes, conditions, semaphores, monitors, and so on, to enforce correct behavior. In the correlator, if you choose to, you can use Apama's MemoryStore correlator plug-in to provide shared state for monitor instances that are being concurrently processed. Other than using the MemoryStore plug-in, data sharing happens only by sending events between monitors.

Apama Architecture

3 Apama Concepts

This section discusses the concepts that are central to all Apama applications. A thorough understanding of these concepts can help you design and develop more robust Apama applications.

Event-driven programming

Events are data elements. Each event is a collection of attribute-value pairs that capture the state (or changes to state) of real-world or computer-based objects. Events consist of data and temporal attributes that represent the *what*, *when*, and *where* of an object. This can be the state of an object or the interaction of objects at a particular time. Real world examples of events include:

- Stock market trades and quotes
- RFID signals
- Satellite telemetry data
- Card swipes at a turnstile
- ATM transactions
- Network activities/faults
- Troop movement on a battlefield
- Activity on a website
- Electronic funds transfers
- SCADA alerts (Supervisory Control and Data Acquisition)

Processing events requires event-driven programming. The hallmarks of event-driven programming include the following:

- Program execution does not flow sequentially from beginning to end. There is no standard starting point.
- Program execution happens in response to the arrival of events. Some external source pushes the events into your program.
- Events arrive in asynchronous messages.
- There are two main bodies of code: code that analyzes incoming events to determine if the events are of interest and code that performs actions when events of interest are found.

There are a lot of similarities between GUI programming and event driven programming. For example, in a GUI program you typically write code that responds to mouse clicks.

See also *Developing Apama Applications in EPL*, "How EPL applications compare to applications in other languages".

Apama Concepts

Complex event processing

Complex Event Processing (CEP) is software technology that enables the detection and processing of

- Events derived from other events — a derived event is an event that your application generates as a result of applying a method or action to one or more other events.
- Event sequences, often with temporal constraints.

CEP programs find patterns in event data that enable detection of opportunities and threats. Timely responses are then pushed to the appropriate recipients. The responses can be in the form of automated events, such as placing orders in algorithmic trading systems, or alerts to someone using Business Activity Monitoring (BAM) dashboards. The result is faster and better operational decisions

EPL and Java provide the features needed to write monitors that can perform CEP. The following example shows how EPL can concisely define event patterns and rules. The `NewsCorrelation` monitor's `onload()` action defines a listener that specifies a complex event expression. The literal translation of the expression is "look for all news articles about any stock, followed by a 5% rise in the value of that stock within 5 minutes". This is the kind of implied news impact that might be of interest to a trader or a market risk analyst.

```
monitor NewsCorrelation {
   NewsItem news;
   StockTick tick;
   action onload {
      on all NewsItem():news {
         on StockTick(symbol=news.subject):tick {
            on StockTick(symbol=news.subject,
                         price >= (tick.price*1.05))
               within(300.0) alertUser;
         }
      }
   }
   action alertUser {
      log "News to price movement Correlation for stock "
                  +news.subject+" has occurred";
   }
}
```

The `on` keyword specifies a listener. The initial listener nests two additional listeners that define the event sequence of interest. The listeners do the following:

1. The initial listener watches for all `NewsItem` events.
2. Each time the correlator detects a `NewsItem` event, this listener captures it in a `news` variable.
3. The first nested listener then watches for a `StockTick` event for the stock that the news item was about. This listener uses the `news` variable to access the information from the previously detected event.
4. When the correlator detects a matching `StockTick` event, the first nested listener captures it in the `tick` variable.
5. The innermost listener then watches for another `StockTick` event for the same stock but with a price that is at least 5% higher than the price in the event captured by the `tick` variable. The `within` keyword indicates that the correlator must detect the second `StockTick` event within 300 seconds (5 minutes) of finding the initial `NewsItem` event.
6. If the correlator finds a second `StockTick` event that matches within 5 minutes, the monitor sends a message to the log file. The nested listeners terminate.

   If the correlator does not find a second `StockTick` event that matches within the 5 minutes, the nested listeners terminate without sending a message to the log.

Apama Concepts

Understanding monitors and listeners

An introduction to monitors and listeners is in "Description of event processing languages" on page 24. As mentioned there, monitors are the basic program component that you inject into the correlator. You write monitors in EPL or Java. If you use Event Modeler to create scenarios, Apama Studio translates your scenario into one or more monitors.

A monitor defines:

- One or more listeners — a listener is the EPL mechanism that specifies the event or sequence of events that you are interested in. Conceptually, listeners sift through the streams of events that come in to the correlator and detect matching events.
- One or more actions — an action is one or more operations that the correlator performs. An action might be the registration of a listener or it might be the execution of an operation when the correlator finds a match between an incoming event or sequence and a listener.

When the correlator executes an `on` statement, it creates a listener. A listener watches for an event, or a sequence of events, that matches the event expression specified in the `on` statement. An event expression defines one or more event templates. Each event template defines an event type to look for, and specifies whether the event's fields should have any specific values. In addition, listeners can specify

- Temporal constraints — for example, a listener can specify that two events of interest must be received within 10 minutes.
- Logic — for example, a listener can specify that it is interested in event `A` or event `B` or event `C`.

It is often desirable to listen, separately but concurrently, for different instances of the same event type. For example, you might want to listen for and process, separately but concurrently, stock tick events for different stocks. EPL accomplishes this by letting a monitor instance spawn other monitor instances.

In the monitor code, you spawn a monitor instance by specifying the `spawn` keyword followed by an action. Each act of spawning creates a new instance of the monitor.

When the correlator spawns a monitor instance, it does the following:

1. The correlator creates a new monitor instance from the original monitor instance. The new monitor instance is almost identical to the original. The new monitor instance has a copy of the variables from the original but the active listeners from the original monitor instance are not copied.
2. The correlator invokes the named action on the new monitor instance.

Monitors that contain a `spawn` statement(s) typically act as factories, creating new monitor instances that all listen for the same event type but where each listens for events that have different values in one or more fields. Also, monitors can spawn to particular threads, referred to as *contexts* in EPL. This enables the correlator to concurrently process multiple monitor instances. (You must create contexts in EPL or use Event Modeler to implement parallel processing. You can refer to contexts from both EPL and Java.)

The lifecycle of a monitor is as follows:

1. You use Apama Developer Studio or a correlator utility to inject the EPL or Java that defines the monitor into the correlator.
2. The correlator creates the original monitor instance, including space for variables as needed.
3. The correlator executes the monitor instance's `onload()` action.
4. The original monitor instance might spawn several times creating new monitor instances. For each spawned monitor instance, the correlator creates a copy of the original monitor instance's variable space and then executes the specified action.
5. A monitor instance terminates when it has no active listeners. Upon termination, the correlator invokes the monitor instance's `ondie()` method, if one is defined. Note that it is possible for a monitor instance to remain active after the monitor instance that spawned it has terminated.
6. When the last instance of a particular monitor terminates, the correlator calls the monitor's `onunload()` method, if it defines one. The last monitor instance to terminate might be the original monitor instance or a spawned monitor instance. Regardless, when the last instance terminates the correlator invokes the monitor's `ondie()` method and then the monitor's `onunload()` method, if these methods are defined.

   For example, suppose that a monitor definition specifies an `ondie()` method and an `onunload()` method. You inject this monitor and the correlator creates the original monitor instance. The original monitor instance spawns 9 times. Consequently, there are 10 instances of that monitor in the correlator. After all of these monitor instances have terminated, the correlator will have called `ondie()` 10 times and it will have called `onunload()` once.

See "Getting started with Apama EPL" in *Developing Apama Applications in EPL*.

Apama Concepts

Understanding scenarios and blocks

A scenario is a real-time, business strategy, involving multi-stage event analysis and action response. You can construct an event-based application from a single scenario or from several cooperating scenarios. At any time, you can add scenarios to the correlator, modify scenarios in the correlator, or remove scenarios from the correlator. You use Event Modeler to define scenarios.

Apama Studio's Event Modeler editor provides the ability for less technical users to quickly create Apama applications in a graphical modeling environment rather than writing event processing logic directly in EPL or Java.

Examples of complex operations that you can implement in scenarios:

- Spotting key events, such as events that are unusual or events that cross a threshold. For example, reporting when a stock price goes above the monthly peak.
- Performing analytics on events or collections of events, to generate derived data that you can analyze. For example, continuously generating a 5-minute moving average for Microsoft's stock price.
- Comparing or correlating values from events over time to detect trends. For example, detecting when the stock price for GE is 10% above the 5-minute moving average.

- Combining time-ordered key events in business patterns that indicate a complex phenomenon has been detected or a business process has completed. For example, spotting that a particular news bulletin has caused Ford's stock price to move by 5%.
- Taking relevant action in response to a detected pattern. For example, automatically buying or selling stock in response to detecting a favorable opportunity.

Each scenario is a template for the instantiation of scenario instances. A good metaphor from object-oriented design is that the scenario is like a class and the scenario instance is like an object; that is, it is an instantiation of the class. The scenario itself can be a generic template, without the need to define the specific values on which to operate. A scenario instance is the running application with all the specific variables and values having been instantiated on creation.

Each scenario has the following components:

- Smart blocks — also referred to as simply blocks. Blocks are useful, reusable, building-block components that generate and/or respond to events. Any scenario can make use of one or more standard or custom blocks. Event Modeler provides a panel for selecting a block you want to use in your scenario.
- Rules — Scenario rules are configurable, if-then statements that interact with smart blocks to create application-specific event logic. Rules specify operations to perform and determine when control transitions from one state to another.
- States — States let a scenario move through different stages, to be able to handle more complex logic. In each state, a set of blocks and rules are active. If a rule causes control to move to a new state then a new set of blocks and rules become active.
- Variables — Each scenario instance uses a set of variables that you specify. You can identify individual scenario variables as being input variables or output variables. To create a scenario instance, you typically need to provide values for a number of input variables. Output variables let you observe the state of the scenario instance.

After you define a scenario, you use Apama Studio or correlator utilities to inject the scenario into the correlator. Once the scenario is in the correlator, you can create instances of it. During development, to create, modify, and delete scenario instances, you can use Apama Studio's Scenario Browser or the scenario's associated dashboard. When a scenario is in production, you use the scenario's dashboard.

Any number of scenario instances can exist at the same time in the correlator. Each instance performs work that is distinct from the work done by the other instances. For example, consider a scenario that specifies a financial trading strategy. Each time someone creates an instance of this scenario, that user can define the instrument to be traded and the criteria for performing the trade.

By default the correlator finishes processing an event for one scenario instance before it begins processing that event for another scenario instance. But you can specify that the correlator should process events for a given scenario's instances in parallel with the processing of events for other scenario instances. For many applications, enabling parallel processing for a scenario improves application performance.

See "Overview of using Event Modeler" in *Developing Apama Applications in Event Modeler*.

Apama blocks are encapsulated elements of CEP logic that you can use in scenario rules. For example, a block might represent:

- A computationally-intensive calculation within an algorithmic trading application

- A complex event pattern to be sought
- The connectivity that integrates Apama with a market data feed.

Blocks make the incorporation of such operations accessible to, not just power-users and programmers, but also business users.

Apama provides a number of standard blocks and these are documented in *Developing Apama Applications in Event Modeler*, "Using standard blocks". Apama Studio provides a tool for defining custom blocks. When defining your scenario, you can add one or more copies of one or more blocks to the scenario. After you add a block to a scenario, you can use the block's parameters, input feeds, operations, and output feeds in the scenario's rules. When you add a block to a scenario, you are effectively specifying that instances of that scenario should each create an instance of that block running within them.

A number of sample scenarios are available in the `samples\scenarios` directory of your Apama installation directory. Examining sample scenarios is a good way to learn typical scenario construction. In addition, all of Apama Studio's sample Apama applications use scenarios, so they also provide good examples from which to learn.

Apama Concepts

Understanding dashboards

A scenario is a representation of application logic, but without any defined user interaction. You add a dashboard to a scenario to enable end-users to:

- Create a new scenario instance. This might include entering the initialization values for the scenario.
- Monitor the status of all scenario instances. For example, to see when a pattern has been detected, and some action taken.
- Manually intervene in the execution of a scenario instance. For example, to take some action in response to an alert.
- Change the configuration of a running scenario instance.
- Deactivate a scenario instance.

In an Apama application, a dashboard is a a real-time, business cockpit for controlling and receiving real-time updates from running scenarios. Deployed dashboards connect to one or more correlators through a dashboard data server. As the scenarios (or data views — see below) in a correlator run, and their variables or fields change, the correlator sends update events to all connected dashboards. When a dashboard receives an update event, it updates its display in real time to show the behavior of the scenario (or data view). User interactions with the dashboard, such as creating an instance of a scenario, result in control events that the dashboard data server sends to the correlator.

See the introduction in *Building Dashboards*.

If your application uses EPL rather than (or in addition to) scenarios you can create data views, which are table structures that specify event fields that you choose. You can then use these data views to create dashboards.

See "Making event type definitons available to monitors" in *Developing Apama Applications in EPL.*

Alternatively, you can use the MemoryStore correlator plug-in in EPL applications. The MemoryStore creates data views for you.

Apama Concepts

4 Getting Ready to Develop Apama Applications

The discussions in the following topics provide a foundation for developing your Apama application.

Becoming familiar with Apama

To become familiar with Apama, you should

- Work through the tutorials in Apama Studio. Select Tutorials from the Apama Studio Welcome page. The tutorials provide step-by-step instructions for developing scenarios in Apama Studio's Event Modeler editor, or EPL or Java applications in Apama Studio's code editors.
- Execute and examine the demonstration applications available from Apama Studio. Select Samples from the Apama Studio Welcome page. The demonstration applications are interactive. You can create instances of scenarios, set parameters for the scenarios, and watch the scenarios execute. The demonstrations provide simple examples of what Apama can do and how you might interact with your Apama application.
- Examine sample code. Your Apama installation directory contains a `samples` directory that contains many examples of EPL programs, Java programs, scenarios, correlator plug-ins, Apama client programs, and more.
- Read all of this material, *Introduction to Apama*, so that you have a broad understanding of what Apama is all about.
- Understand what is covered in the Apama user documentation. Peruse the documentation so that you know where to look for particular information. You can then refer to the documentation for the component you need to use.

Getting Ready to Develop Apama Applications

Introduction to Apama Studio

Apama Studio is the main tool for implementing Apama applications. Apama Studio is a set of Eclipse plug-ins that provides a number of Eclipse perspectives:

- Use the Apama Workbench perspective when you are new to Apama. This perspective provides a simplified view of Apama features that makes it easy to get started developing Apama applications.
- You can use Event Modeler to develop scenarios in either the Workbench or Developer perspective.
- Use the Apama Developer perspective when you are comfortable using the Apama Workbench perspective. The Developer perspective gives you far more control over your Apama project than the Apama Workbench perspective. For example, you can view more than one Apama project at one time, and you can specify launch configuration parameters.
- Use the Apama Runtime perspective for monitoring and debugging the execution of Apama applications.
- Use the Apama Debug perspective to debug your Apama application. The Debug perspective allows you to set break points, examine variable values, and control execution.

- Use the Apama Profiler perspective to profile your Apama application. The Profiler perspective allows you to see which components of your application are consuming the most CPU time or to see if there are other bottlenecks in the application.

When developing an Apama application, the first step is to create an Apama project to contain your application files. An Apama project is a convenient way to manage the various files that make up your application. For example, an Apama application can include the following types of files:

- EPL files (`.mon` extension)
- Scenario definition files (`.sdf` extension)
- Block definition files (`.bdf` extension)
- Java files
- Dashboard files (`.rtv` extension)
- Files that contain sample events (`.evt` extensions)
- C, C++, Java and .NET files that contain Apama client applications or correlator plug-ins
- Adapters that provide the interface between your event sources and Apama
- Image files for your dashboards
- Text, HTML or XML files

You can add and manage all of these files from your Apama project in r Studio. In addition, Apama Studio provides EPL and Apama Java editors whose features include content assistance, auto-bracketing, templates for frequently entered constructs, and problem detection. After you build an Apama project, Apama Studio flags any line that contains an error.

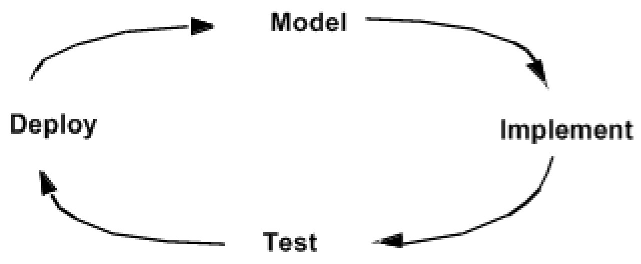
If your project contains dashboards and scenarios, Apama Studio opens the Dashboard Builder or Event Modeler when you double-click `.rtv` or `.sdf` files, respectively. You can also use Apama Studio to test your application. Apama Studio provides features that inject your application into the correlator, send test event streams to the correlator, launch adapters, and configure and monitor the operation of your application in a test environment.

Finally, Apama Studio provides tools for packaging your application so that you can deploy it. See "Overview of Apama Studio" in *Using Apama Studio*.

Getting Ready to Develop Apama Applications

Steps for developing Apama applications

Typically, Apama development is an iterative cycle:



Multiple contributors with varying expertise can work concurrently to develop an Apama application.

The main steps for developing an Apama application include:

1. Model: Design your application. Important tasks are modeling the events that your application needs to handle and identifying the services that your application must provide.
2. Implement: Use Apama Studio to create an Apama project to contain your application files (EPL files, adapters, event files, scenarios, dashboards, and so on). Since Apama applications typically consist of many components, it is often possible to concurrently implement them, particularly if several people are working on the application:
   - Create scenarios in Apama Studio's Event Modeler.
   - Write EPL programs or Apama Java programs in Apama Studio.
   - Develop Apama client applications.
   - Implement or develop adapters.
   - Create dashboards in Dashboard Builder.
   - Develop correlator plug-ins that extend the correlator's standard features.
3. Test: Apama Studio provides a runtime perspective and Scenario Browser view that help test applications as they are built. You can also use Apama Studio's Data Player in conjunction with the ADBC adapter to analyze application behavior before, or after, deployment. You can automate testing through the use of command-line clients.
4. Deploy: Use the Management and Monitoring console to start and manage Apama components, including correlators. Manage access to Apama applications with dashboards. Collect and manage event data with the ADBC (Apama Database Connector) standard Apama adapter and Apama Studio's Data Player. Tune Apama applications for optimum performance.

   See "Overview of deploying Apama applications" in *Deploying and Managing Apama Applications*.

Getting Ready to Develop Apama Applications

Overview of starting, testing and debugging applications

Apama Studio provides tools for running your Apama application in a test environment.

In the Apama Workbench perspective, click the Start button to start a correlator and inject the current project. Apama Studio then displays the Scenario Browser panel. Use the Scenario Browser to create running instances of your scenarios and examine parameter values during execution. You can monitor execution in the Console and Problems panes.

In the Apama Developer perspective, select the project you want to test. In the Apama Studio menu bar, select Run and then select whether you want to run, debug or profile your Apama application. You can specify one or more launch configurations for your project.

In the Apama Runtime perspective, you can monitor your running application.

See *Using Apama Studio*, "Launching projects", "Debugging EPL applications", and "Debugging Apama Java applications".

Getting Ready to Develop Apama Applications

Apama Glossary

action

A component of a monitor. An action specifies a set of operations to perform if there is a match between an incoming event and a listener’s event template.

activation

When the passage of time or the arrival of an item causes a stream network or an element in a stream network to process items.

adapter

Software component that translates events from a non-Apama format to Apama format. This allows the correlator to analyze the event. An adapter plugs into the Apama Integration Adapter Framework (IAF) and injects events into the correlator. Adapters can be bi-directional converting event formats in both directions.

aggregate function

A function that operates on all items in a query window, for example, `sum()`.

Apama Studio

Eclipse-based GUI for managing Apama projects, developing EPL files, developing custom blocks, and running Apama applications in test environments.

batch

When you define a window in a stream query you can specify that you want to update the window in batches. A batch can be a certain number of items or it can be the items that arrived in a certain length of time.

block

Reusable module that generates or responds to events. You can import and use one or more blocks in a scenario. Blocks can accept input, execute some logic, and return some results. Blocks can consist of input feeds, output feeds, parameters, and operations. Apama provides standard blocks, which are described in *Developing Apama Applications in Event Modeler*, "Using standard blocks" (available if you selected Developer during installation). You can define additional blocks by using Apama Studio’s block editor. Also called a SmartBlock.

bundle

In Apama Studio, a bundle is a named collection of Apama-provided objects that are required to execute a particular type of Apama application. Typically, a bundle includes EPL files, event definition files and event files, but it can include a wide range of file types such as IAF configuration files. For example, when your project includes a scenario, you add the `Scenario Service` bundle to your project. Your application then has everything it needs to execute a scenario.

.cdp

File extension ("correlator deployment package") for Apama correlator deployment packages.

CEP

Complex event processing. CEP technologies let you detect and process events derived from other events, and sequences of events with or without temporal constraints.

context

Contexts allows EPL applications to organize work into threads that the correlator can concurrently execute. In EPL, `context` is a reference data type. When you create a variable of type `context`, or an event field of type `context`. you are actually creating an object that refers to a context. The context might or might not already exist. You can then use the context reference to spawn to the context or enqueue an event to the context. When you spawn to a context, the correlator creates the context if it does not already exist.

correlator

Event correlation engine. The part of Apama that looks for events of interest, analyses matching events, and executes appropriate actions.

correlator deployment package

A correlator deployment package (CDP) is a file that contains application EPL code in a proprietary, non-plain-text format. These files treat EPL files similarly to the way Java files are treated in JAR files. CDP files can be created by Exporting from Apama Studio projects or by using the `engine_package` utility. CDP files can be injected to the correlator just as EPL files and JAR files containing Apama Java applications are injected.

correlator-integrated JMS messaging

Apama's correlator-integrated JMS messaging provides an efficient way for Apama applications to send messages and to receive JMS messages for processing. Correlator-integrated JMS messaging also provides for reliable messaging (guaranteed delivery) and duplicate detection.

.csv

File extension ("comma separated values") for some exported data; suitable for third party applications such as spread sheets.

dashboard

Business cockpit for controlling, receiving, and visualizing real-time updates from running scenarios.

Dashboard Builder

GUI for creating and modifying dashboards.

dashboard data server

Process that mediates communication between dashboards and running scenarios.

Dashboard Viewer

Desktop application that supports local deployment of dashboards.

Data Player

Apama Studio component that lets you retrieve events that pass through the correlator. You can use the Data Player to play back stored events and use the results to develop, test, and debug scenarios.

data view

Table structure that contains event fields that you specify. In EPL applications, you create data views so that you can use the Dashboard Builder to create dashboards that let you interact with your running EPL application in the correlator.

.ddf

File extension for dashboard description files.

.ddp

File extension for deployment description package files.

EDA

Event-driven architecture. Architecture designed to take in tens of thousands of events from multiple feeds, analyze those events to detect the events of interest, and execute actions in response to relevant events.

EPL

Apama Event Processing Language (EPL) is an event-based scripting language that is an interface to the correlator. Java is the other interface to the correlator. EPL is the new name for MonitorScript. Within the product, both EPL and Monitorscript are used and should be treated as synonymous.

event

An occurrence of a particular circumstance of interest at a specific time that usually corresponds to a message of some form. The message is a collection of attribute-value pairs that describe a change in an object.

event collection

The process of storing events that stream through the correlator. The collected events can be played back (with Apama Studio's Data Player) to analyze what happened or to test alternative scenario strategies. The collected events can also be exported to spreadsheet applications.

event listener

An event listener observes the correlator event stream analyzing each event in turn until it finds a sequence of events that match its event expression. When this happens the event listener triggers, causing the correlator to execute the listener action. See also *stream listener*.

Event Modeler

Apama Studio editor that you use to define scenarios.

event template

Basic unit of monitoring in the correlator. An event template specifies the pattern that you want to act on. A simple scenario contains one or a few event templates. A more complex scenario can contain many event templates. Here is an example of the data that a particular event template might define:

- Instrument = IBM
- Bid Price > 93 and < 95
- Offer Price = *
- Bid Volume > 100000
- Offer Volume = *

.evt

File extension for files that contain events.

exception

An exception is an object that represents a runtime error that can be caught with a try-catch statement. In EPL, `Exception` is a reference data type in the `com.apama.exceptions` namespace. See "Catching exceptions" in *Developing Apama Applications in EPL*.

IAF

Integration Adapter Framework.

Integration Adapter Framework (IAF)

Server component that adapters plug into for runtime invocation.

JMON

Apama in-process Java API (Java MONitors). This term is deprecated.

listener

See *event listene*r and *stream listener*.

lot

The items produced by a single activation of a stream query. Like an auction lot, a stream query lot can contain one or more items.

Management and Monitoring

An Apama service that enables centralized configuration of an Apama deployment and uses a health protocol to monitor all components in an Apama deployment. A management and monitoring console lets you configure and assess the status of the running system.

.mon

File extension for EPL files.

monitor

A monitor contains event monitoring patterns and the responses to take when the monitor's listeners detect those patterns. You can use EPL or Java to define a monitor.

MonitorScript

Apama event-based scripting language that is an interface to the correlator. Java is the other interface to the correlator. EPL is the new name for MonitorScript. Within the product, both EPL and Monitorscript are used and should be treated as synonymous.

partitioning

A strategy to scale Apama by deploying multiple correlator processes to spread the workload across several processors and/or machines. A correlator can be used to partion incoming events, sending them to different correlators based on rules specific to your partioning strategy.

plug-in

Plug-ins are C or C++ code modules that you write to extend the capability of an Apama component. Apama provides APIs that let you write plug-ins for correlators, dashboards, and adapters.

.rtv

File extension for dashboard view files.

scenario

An independent real-time business strategy that contains a number of states. You use Apama Studio's Event Modeler editor to create a scenario. You inject a scenario into the correlator. You then create an instance of the scenario, which listens for a particular event. When that event occurs, the scenario performs specified actions according to the rules defined in the scenario.

.sdf

File extension for scenario definition files.

sentinel agent

A small process that starts and stops Apama correlators and IAF adapters on the host where it is running. A sentinel agent is necessary so that the Management and Monitoring console can start and stop components on that host.

simulation

A Data Player playback session that uses persisted event data for "what if" analysis. A simulation can test what would happen with modified data or what would happen with a modified scenario.

SmartBlock

See *block*.

stack trace element

A stack trace element is an object that describes an entry in the stack trace. A `com.apama.exceptions.Exception` object contains a sequence of stack trace elements that show where an exception was first thrown and the calls that lead to that exception. In EPL, `com.apama.exceptions.StackTraceElement` is a reference data type. See .

standard blocks

Blocks provided with Apama. You can use Apama Studio's block editor to write additional blocks. See also *block*.

stream

A conduit or channel through which items flow. An item can be an event, a `location` type or a simple type (`boolean`, `decimal`, `float`, `integer`, or `string`). The set of items flowing through the stream is often referred to as *a stream of items* and so, here, a stream represents *an ordered sequence of items over time*. A stream transports items of only one type. Streams are internal to a monitor.

stream listener

A construct that continuously watches for items from a stream and invokes the listener code block each time new items are available.

stream network

A network of stream source templates, streams, stream queries, and stream listeners. The upstream elements in the stream network feed the downstream elements to generate derived, added-value items.

stream source template

An event template preceded by the `all` keyword. It uses no other event operators. A stream source template creates a stream that contains events that match the event template.

stream query

A query that the correlator applies continuously to one or two streams. The output of a stream query is one continuous stream of derived items.

Web Services Client adapter

The Apama Web Services Client adapter is usable across all Apama components. Interaction with the IAF is transparent to the application developer. See also *adapter*.

window

A dynamic portion of the items flowing through a stream. A window identifies which items a stream query is currently processing.