

# Deploying and Managing Apama Applications

Version 9.10

August 2016

This document applies to Apama Version 9.10 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 © 2013-2016 Software AG, Darmstadt, Germany and/or Software AG USA Inc., Reston, VA, USA, and/or its subsidiaries and/or its affiliates 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://softwareag.com/licenses>.

Use of this software is subject to adherence to Software AG's licensing conditions and terms. These terms are part of the product documentation, located at <http://softwareag.com/licenses> 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, license terms, additional rights or restrictions, please refer to "License Texts, Copyright Notices and Disclaimers of Third Party Products". For certain specific third-party license restrictions, please refer to section E of the Legal Notices available under "License Terms and Conditions for Use of Software AG Products / Copyright and Trademark Notices of Software AG Products". These documents are part of the product documentation, located at <http://softwareag.com/licenses> and/or in the root installation directory of the licensed product(s).

# Table of Contents

<b>About this Guide.....</b>	<b>7</b>
Documentation roadmap.....	7
Online Information.....	9
Contacting customer support.....	10
<b>Security Requirements for Apama.....</b>	<b>11</b>
<b>Overview of Deploying Apama Applications.....</b>	<b>13</b>
About deploying components with Command Central.....	14
Getting started with Command Central.....	14
Deploying an Apama project in Command Central.....	22
About deploying Apama applications with a YAML configuration file.....	23
About deploying components with EMM.....	24
About Apama command line utilities.....	24
About deploying dashboards.....	24
About tuning applications for performance.....	25
Setting up the environment using the Apama Command Prompt.....	26
<b>Using the Management and Monitoring Console.....</b>	<b>27</b>
Installation.....	28
Concepts.....	29
The EMM console window.....	30
Committed values and outstanding changes.....	31
State persistence.....	32
Managing licenses.....	33
Managing security.....	34
Managing hosts.....	35
Sentinel Agents.....	35
Using the sentinel_agent tool.....	36
Working directory.....	38
Working with hosts.....	38
Managing all known hosts.....	44
Managing components.....	45
Component status indicators.....	46
Working with components.....	47
Add correlator.....	47
Add adapter.....	47
Remove component.....	48
Move component to host.....	48
Start component.....	49
Stop component.....	50
Restart component.....	50

Clear all component logs.....	50
Configuring components with the details pane.....	51
Specifying paths and filenames in the Details Pane.....	51
Preferences.....	52
Warnings.....	52
Display.....	55
Timing.....	57
Deploying and Configuring Correlators.....	58
Adding correlators.....	59
The correlator tabs.....	60
Management tab.....	60
Configuration tab.....	61
Initialization tab.....	63
Adding initialization actions.....	64
Connections tab.....	67
Adding new upstream connections.....	68
Downstream connections.....	70
Statistics tab.....	71
Inspect tab.....	74
Deleting Monitors, Event Types, or JMon applications from a correlator.....	76
Displaying event type definition for multiple Event Types.....	76
Diagnostics tab.....	77
Deploying and Configuring Adapters.....	79
Adding adapters.....	79
Configuring adapters.....	80
Specifying paths and filenames in the Details Pane.....	80
The Adapter tabs.....	81
Management tab.....	81
Configuration tab.....	82
Connections tab.....	84
Control tab.....	84
Statistics tab.....	84
Diagnostics tab.....	88
<b>Deploying and Managing Queries.....</b>	<b>91</b>
Overview of deploying and managing query applications.....	92
Query application architecture.....	92
Deploying query applications.....	93
Running queries on correlator clusters.....	94
Deploying queries on multiple correlators.....	94
Deploying BigMemory Terracotta Server Array.....	95
Configuring BigMemory driver.....	95
Using JMS to deliver events to queries running on a cluster.....	96
Mixing queries with monitors and scenarios.....	97
Handling node failure and failover.....	98

Managing parameterized query instances.....	98
Creating new query instances by setting parameter values.....	99
Changing parameter values for queries that are running.....	99
Monitoring running queries.....	99
<b>Tuning Correlator Performance.....</b>	<b>103</b>
Scaling up Apama.....	104
Partitioning strategies.....	105
Engine topologies.....	109
Event correlator pipelining.....	111
Configuring pipelining with engine_connect.....	111
Connection configuration file.....	117
Configuring pipelining through the client API.....	118
Event partitioning.....	118
<b>Managing and Monitoring over REST.....</b>	<b>121</b>
Generic Management.....	125
Correlator Management.....	126
IAF Management.....	127
Dashboard Management.....	127
<b>Correlator Utilities Reference.....</b>	<b>129</b>
Starting the event correlator.....	130
Specifying log filenames.....	139
Examples for specifying log filenames.....	141
Descriptions of correlator status log fields.....	143
Text internationalization issues.....	146
Determining whether to disconnect slow receivers.....	146
Description of slow receivers.....	146
How frequently slow receivers occur.....	147
Correlator behavior when there is a slow receiver.....	147
Tradeoffs for disconnecting a slow receiver.....	148
Determining whether to disable the correlator's internal clock.....	148
Injection time of compiled runtime.....	149
Injecting code into a correlator.....	150
Deleting code from a correlator.....	153
Packaging EPL and Java code.....	156
Sending events to correlators.....	158
Receiving events from correlators.....	162
Watching correlator runtime status.....	164
Inspecting correlator state.....	170
Shutting down and managing components.....	172
Viewing garbage collection information.....	182
Using the EPL memory profiler.....	182
Using the CPU profiler.....	189
Setting logging attributes for packages, monitors and events.....	194

Rotating the correlator log file.....	197
Rotating all log files.....	198
Rotating specified log files.....	198
Using the command-line debugger.....	199
Obtaining online help for the command-line debugger.....	205
Enabling and disabling debugging in the correlator.....	205
Working with breakpoints using the command-line debugger.....	205
Controlling execution with the command-line debugger.....	206
The wait command.....	207
Command shortcuts for the command-line debugger.....	207
Examining the stack with the command-line debugger.....	209
Displaying variables with the command-line debugger.....	209
Generating code coverage information about EPL files.....	209
Recording code coverage information.....	210
Creating code coverage reports.....	212
Interpreting the HTML code coverage reports.....	214
Using EPL code coverage with PySys tests.....	215
Replaying an input log to diagnose problems.....	216
Creating an input log file.....	216
Rotating an input log file.....	216
Performance when generating an input log.....	217
Reproducing correlator behavior from an input log.....	217
Event file format.....	219
Event representation.....	220
Event timing.....	220
Event types.....	222
Event association with a channel.....	222
Using the data player command-line interface.....	223
Using the Apama component extended configuration file.....	225
Binding server components to particular addresses.....	226
Ensuring that client connections are from particular addresses.....	226
Setting environment variables for Apama components.....	227
Setting log files and log levels in an extended configuration file.....	227
Sample extended configuration file.....	229
Starting a correlator with an extended configuration file.....	229

## About this Guide

---

*Deploying and Managing Apama Applications* discusses the following topics:

- Issues involved in deploying the components that make up an Apama application
- Starting and managing Apama correlators and adapters with the Enterprise Management and Monitoring console (EMM)
- Tuning Apama applications for optimum performance
- Managing Apama applications over REST
- Using correlator utilities and the extended component configuration file

## Documentation roadmap

---

Apama provides documentation in the following formats:

- HTML (viewable in a web browser)
- PDF (available from the documentation website)
- Eclipse help (accessible from the Software AG Designer)

You can access the HTML documentation on your machine after Apama has been installed:

- **Windows.** Select **Start > All Programs > Software AG > Tools > Apama *n.n* > Apama Documentation *n.n***. Note that **Software AG** is the default group name that can be changed during the installation.
- **UNIX.** Display the `index.html` file, which is in the `doc` directory of your Apama installation directory.

The following table describes the different guides that are available.

Title	Description
<i>Release Notes</i>	Describes new features and changes since the previous release.
<i>Installing Apama</i>	Instructions for installing Apama.
<i>Introduction to Apama</i>	Introduction to developing Apama applications, discussions of Apama architecture and concepts, and pointers to sources of information outside the documentation set.

Title	Description
<i>Using Apama with Software AG Designer</i>	Instructions for using Apama to create and test Apama projects, develop EPL programs, define Apama queries, develop JMon programs, and store, retrieve and play back data.
<i>Developing Apama Applications</i>	Describes the different technologies for developing applications: EPL monitors, Apama queries, Event Modeler, and Java. You can use one or several of these technologies to implement a single Apama application. In addition, there are C++, C, and Java APIs for developing components that plug in to a correlator. You can use these components from EPL.
<i>Connecting Apama Applications to External Components</i>	<p>Describes how to connect Apama applications to any event data source, database, messaging infrastructure, or application. The general alternatives for doing this are as follows:</p> <ul style="list-style-type: none"><li>■ Implement standard Apama Integration Adapter Framework (IAF) adapters.</li><li>■ Create applications that use correlator-integrated messaging for JMS or Software AG's Universal Messaging.</li><li>■ Use connectivity plug-ins written in Java or C++.</li><li>■ Develop adapters with Apama APIs for Java and C++.</li><li>■ Develop client applications with Apama APIs for Java, .NET, and C++.</li></ul>
<i>Building and Using Dashboards</i>	<p>Describes how to build and use an Apama dashboard, which provides the ability to view and interact with scenarios and DataViews. An Apama project typically uses one or more dashboards, which are created in the Dashboard Builder. The Dashboard Viewer provides the ability to use dashboards created in Dashboard Builder. Dashboards can also be deployed as simple Web pages, applets, or WebStart applications. Deployed dashboards connect to one or more correlators by means of a Dashboard Data Server or Display Server.</p>



Title	Description
<i>Deploying and Managing Apama Applications</i>	<p>Describes how to deploy components with Command Central or with Apama's Enterprise Management and Monitoring (EMM) console. Also provides information for:</p> <ul style="list-style-type: none"><li>■ Improving Apama application performance by using multiple correlators and saving and reusing a snapshot of a correlator's state.</li><li>■ Managing and monitoring over REST (Representational State Transfer).</li><li>■ Using correlator utilities.</li></ul>

In addition to the above guides, Apama also provides the following API reference information:

- API Reference for EPL in ApamaDoc format
- API Reference for Java in Javadoc format
- API Reference for C++ in Doxygen format
- API Reference for .NET in HTML format

## Online Information

---

### Software AG Documentation Website

You can find documentation on the Software AG Documentation website at <http://documentation.softwareag.com>. The site requires Empower credentials. If you do not have Empower credentials, you must use the TECHcommunity website.

### Software AG Empower Product Support Website

You can find product information on the Software AG Empower Product Support website at <https://empower.softwareag.com>.

To submit feature/enhancement requests, get information about product availability, and download products, go to [Products](#).

To get information about fixes and to read early warnings, technical papers, and knowledge base articles, go to the [Knowledge Center](#).

### Software AG TECHcommunity

You can find documentation and other technical information on the Software AG TECHcommunity website at <http://techcommunity.softwareag.com>. You can:

- Access product documentation, if you have TECHcommunity credentials. If you do not, you will need to register and specify "Documentation" as an area of interest.
- Access articles, code samples, demos, and tutorials.
- Use the online discussion forums, moderated by Software AG professionals, to ask questions, discuss best practices, and learn how other customers are using Software AG technology.
- Link to external websites that discuss open standards and web technology.

## Contacting customer support

---

If you have an account, you may open Apama Support Incidents online via the eService section of Empower at <https://empower.softwareag.com/>. If you do not yet have an account, send an email to [empower@softwareag.com](mailto:empower@softwareag.com) with your name, company, and company email address and request an account.

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

# 1 Security Requirements for Apama

---

## Security model

The Apama security model for correlator and IAF components is that untrusted users must not be given access to any related files or to send or receive data on any of the correlator or IAF network ports. It is assumed that any user able to access these files or ports is trusted and has permission to make arbitrary changes and read arbitrary data. Such users are also permitted to inject arbitrary code into a correlator and execute it with the permissions of the correlator process.

## Security requirements

You must use the standard tools and configuration means of your operating system and network to restrict access to the IAF and correlator components to only trusted users.

You must use the standard tools of your operating system to restrict access to all configuration and data files to only trusted users.

Correlator and IAF log levels must be set to `INFO` (or more verbose) to ensure that all security-relevant events are written to the log files.

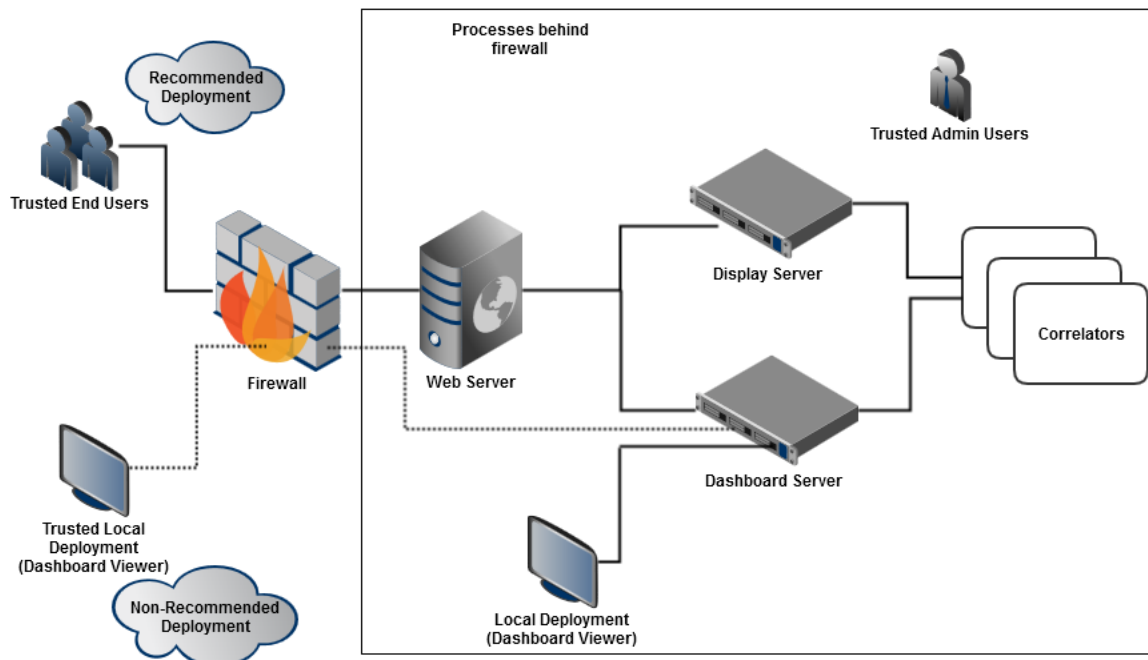
If components must connect across an untrusted network, then either a standard overlay tool such as a VPN must be used, or the interconnection has to be done via Universal Messaging with appropriate controls and restrictions configured in Universal Messaging.

If you need to restrict the data sent from certain users to the correlator, then all interconnections must be done via Universal Messaging with appropriate controls and restrictions configured in Universal Messaging.

## Dashboards

For access from untrusted hosts, you should deploy your dashboards to the web server using either a Display Server, applet and/or WebStart deployment option. The Dashboard Server or Display Server processes should be running behind a firewall, just like the correlators. When accessing the Dashboard Server by using a standalone Dashboard Viewer outside the firewall, make sure to run the Dashboard Server with the `--ssl` option, which will ensure secure sockets for client communication using the data port. You must also ensure that the management ports of both servers are not exposed to end users.

The following diagram depicts the recommended dashboard deployment options:



Authentication for the Display Server is done via JAAS and your authentication mechanism of choice.

**Important:** You must customize your own JAAS modules. The out-of-the-box authentication/authorization modules for dashboards cannot be used. This means that there is no authentication by default, and the basic authorization mechanism is shipped. For details on how to configure your own JAAS modules, see "Administering Dashboard Security" in *Building and Using Dashboards*.

For a dashboard audit trail, you must load the Dashboard Support bundle into the correlator processing audit events and handle the `DashboardClientConnected` and `DashboardClientDisconnected` events, logging them in an appropriate manner.

## 2 Overview of Deploying Apama Applications

---

■ About deploying components with Command Central .....	14
■ About deploying Apama applications with a YAML configuration file .....	23
■ About deploying components with EMM .....	24
■ About Apama command line utilities .....	24
■ About deploying dashboards .....	24
■ About tuning applications for performance .....	25
■ Setting up the environment using the Apama Command Prompt .....	26

Deploying and managing Apama applications involves the following activities:

- Starting and managing Apama components with the Enterprise Management and Monitoring console (EMM)
- Starting and managing correlators with Apama command line utilities
- Tuning Apama applications for optimum performance

## About deploying components with Command Central

---

Software AG Command Central is a tool that release managers, infrastructure engineers, system administrators, and operators can use to perform administrative tasks from a single location. Command Central can assist with the following configuration, management, and monitoring tasks:

- Infrastructure engineers can see at a glance which products and fixes are installed, and where. Engineers can also easily compare installations to find discrepancies.
- System administrators can configure environments using a single web UI, command line tool, or API so maintenance can be performed with a minimum effort of risk.
- Release managers can prepare and deploy changes to multiple servers using command-line scripting for simpler, safer lifecycle management.
- Operators can monitor server status and health, as well as start and stop servers from a single location. They can also configure alerts to be sent to them in case of unplanned outages.

For Apama components, Command Central supports the following features:

- Creating, deleting, starting, stopping, and configuring Apama component instances (correlator, Integration Adapter Framework (IAF), Dashboard Data Server, and Dashboard Display Server instances).
- Monitoring whether component instances are started or stopped.

## Getting started with Command Central

Ensure you have installed both Apama Server (PAMServer) and Apama Platform Manager Plug-in (PAMspm) on every machine on which you wish to start Apama components, and have also installed the Command Central server on at least one machine. For more information, see "Installing Apama Using Command Central" in *Installing Apama*.

- Perform any required post installation tasks as described in "Performing Post Installation Configuration" section in *Software AG Command Central Help*.
- Optional. Define environment variables so that you can invoke Command Central and Platform Manager commands from any location on the machine. To do so:

1. Set the `CC_CLI_HOME` environment variable to the following directory: `Software AG_directory\CommandCentral\client`.

Examples:

- Windows: `set CC_CLI_HOME=C:\SoftwareAG\CommandCentral\client`
- UNIX: `export CC_CLI_HOME="/opt/SoftwareAG/CommandCentral/client"`

2. Add `$CC_CLI_HOME/bin` to the `PATH` environment variable.

Examples:

- Windows: `set PATH=%PATH%;%CC_CLI_HOME%\bin`
- UNIX: `export PATH="$PATH:$CC_CLI_HOME/bin"`

- Define `CC_USERNAME` and `CC_PASSWORD` environment variables to a user name and password. The default username is *Administrator* and password is *manage*.

Examples:

- Windows: `set CC_USERNAME=Administrator`
- Windows: `set CC_PASSWORD=manage`
- UNIX: `export CC_USERNAME="Administrator"`
- UNIX: `export CC_PASSWORD="manage"`

- Open command prompt. Change the directory to `Software AG_directory\CommandCentral\client\bin` (you do not have to change the directory if you have defined `CC_CLI_HOME` environment variable). The `bin` folder contains all the executable files for the Command Central commands.

**Note:** You can create and delete Apama component instances using the command line interface only.

- Most tasks can be performed using the Command Central web interface or command line interface. However, tasks such as instance creation and deletion can only be performed using the command line tool. For more information about using the Command Central command line interface and web user interface, see "Understanding Command Central" in *Software AG Command Central Help*.

The following examples illustrate how the Command Central command line interface can be used to perform some of the most common operations for Apama component instances. These examples assume that you have set the `CC_PASSWORD` environment variable and you are running the command line tool on the machine where Command Central is installed, and configuring the components to run on the Platform Manager node with the default node alias of `local`.

In the following example commands, `<nodeAlias>` is `local`.

`runtimeComponentID` is the ID of an instance in the format *Apama-ComponentType-InstanceName*, where

- *ComponentType* is one of the supported Apama components `correlator`, `iaf`, `displayserver`, or `dataserver`.
- `myCorrelator`, `myIAFmyDisplayServer`, and `myDataServer` are the *InstanceName* when the instance was created.

*configurationTypeId* is one of the supported configuration types:

- APAMA-ARGS
- COMMON-PORTS
- COMMON-LICENSE
- COMMON-LICLOC
- APAMA-ENVVAR
- COMMON-MEMORY
- COMMON-COMPONENT-ENDPOINTS
- Creating and deleting Apama component instances. These operations can be performed using the command line interface only.

- To create an instance of correlator:

```
cc create instances local PAMServer
instance.name=myCorrelator instance.type=correlator instance.port=15993
```

- To create an instance of IAF:

```
cc create instances local PAMServer instance.name=myIAF
instance.type=iaf instance.port=15993 iafConfigFile=iaf-config.xml
```

- To create an instance of Display Server:

```
cc create instances local PAMServer instance.name=myDisplayServer
instance.type=displayserver dataPort=3279
managementPort=28889
```

- To create an instance of Data Server:

```
cc create instances local PAMServer instance.name=myDataServer
instance.type=dataserver dataPort=3278
managementPort=28888
```

- To delete an instance of correlator:

```
cc delete instances local Apama-correlator-myCorrelator
```

After an instance is created, the instance is referred to in the other commands by the component identifier `Apama-instance.type-instance.name`.

- Starting, stopping, and restarting Apama component instances. These operations can be performed using the web interface and the command line interface.

- To start an Apama component:

```
cc exec lifecycle start local Apama-correlator-myCorrelator
```

- To stop an Apama component:

```
cc exec lifecycle stop local Apama-correlator-myCorrelator
```



- To restart an Apama component:

```
cc exec lifecycle restart local Apama-correlator-myCorrelator
```

- Configuring Apama component instances. This operation can be performed using the web interface and the command line interface. You can use the following command to configure an Apama component instance:

```
cc update configuration data <nodeAlias>  
<runtimeComponentId> <configurationInstanceId> -i <Propertiesfile>
```

The format of the properties file is provided below after the example commands.

Use `cc get configuration instances` command to retrieve information about an instance such as the instance ID, the display name for an instance, and the description for an instance. For example, the following command displays the information of a correlator instance.

```
cc get configuration instances local Apama-correlator-myCorrelator
```

Apama supports the following configuration types:

### APAMA-ARGS

- To update APAMA-ARGS configuration type for runtime correlator instance:

```
cc update configuration data local  
Apama-correlator-myCorrelator APAMA-ARGS -i C:\args_data.properties
```

- To update APAMA-ARGS configuration type for runtime IAF instance:

```
cc update configuration data local  
Apama-iaf-myIAF APAMA-ARGS -i C:\args_data.properties
```

- To update APAMA-ARGS configuration type for runtime Display Server instance:

```
cc update configuration data local  
Apama-displayserver-myDisplayServer APAMA-ARGS -i  
C:\args_data.properties
```

- To update APAMA-ARGS configuration type for runtime Data Server instance:

```
cc update configuration data local  
Apama-dataserver-myDataServer APAMA-ARGS -i C:\args_data.properties
```

Example format of a properties file for correlator:

```
extraArgs= -V DEBUG  
logLevel=CRIT  
inputLog=default-correlator-input_testing.log  
outputLog=testing.log  
enableCorrelatorPersistence=true  
persistentDatastorePath=C:\myStore.db  
clearPersistentStateOnStartup=true
```

Example format of a properties file for IAF:

```
extraArgs= -v INFO  
iafConfigFile= C:\iaf-config.xml  
logLevel=DEBUG  
outputLog=C:\output.txt
```

Example format of a properties file for Display Server:

```
extraArgs= -v INFO
logLevel=DEBUG
outputLog=C:\output.txt
connectMode=always
```

Example format of a properties file for a Data Server:

```
extraArgs= -v INFO
logLevel=DEBUG
outputLog=C:\output.txt
connectMode=always
namedServerMode=true
```

## COMMON-PORTS

- To update port configuration for correlator instance:

```
cc update configuration data local Apama-correlator-myCorrelator
COMMON-PORTS-port -i Correlator.xml
```

- To update port configuration for IAF instance:

```
cc update configuration data local Apama-iaf-myIAF
COMMON-PORTS-port -i IAF.xml
```

- To update data port configuration for Display Server instance:

```
cc update configuration data local Apama-displayserver-myDisplayServer
COMMON-PORTS-dataPort -i DataPort_displayServer.xml
```

- To update management port configuration for Display Server instance:

```
cc update configuration data local Apama-displayserver-myDisplayServer
COMMON-PORTS-managementPort -i ManagementPort_displayServer.xml
```

- To update data port configuration for Data Server instance:

```
cc update configuration data local Apama-dataserver-myDataServer
COMMON-PORTS-dataPort -i DataPort_dataServer.xml
```

- To update management port configuration for Data Server instance:

```
cc update configuration data local Apama-dataserver-myDataServer
COMMON-PORTS-managementPort -i ManagementPort_dataServer.xml
```

Example format of XML file for IAF and correlator for the instance ID COMMON-PORTS-port:

```
<PortSettings>
  <Port alias="port">
    <Number>12345</Number>
    <Protocol>HTTP</Protocol>
  </Port>
</PortSettings>
```

Example format of XML file for dashboard management port with the instance ID COMMON-PORTS-managementPort:

```
<PortSettings>
  <Port alias="managementPort">
    <Number>12345</Number>
    <Protocol>HTTP</Protocol>
  </Port>
</PortSettings>
```

Example format of XML file for dashboard data port with the instance ID COMMON-PORTS-dataPort:

```
<PortSettings>
  <Port alias="dataPort">
    <Number>12345</Number>
    <Protocol>HTTP</Protocol>
  </Port>
</PortSettings>
```

## COMMON-LICENSE

- To specify the license key file for a correlator instance:

```
cc update configuration license local Apama-correlator-Correlator1
COMMON-LICENSE-PAMServer-correlator-Correlator1 CorrelatorLicenseAlias
```

In the above example, CorrelatorLicenseAlias is the alias to the license key file.

## COMMON-LICLOC

Applies only to Apama correlator instances. This configuration type contains the location of the license key file. After you set the license key file using COMMON-LICENSE configuration type, the license key file information will be available at APAMA\_HOME/command-central/instances/correlator/<instancename>/ApamaCorrelatorLicense.xml. This license key location is stored in COMMON-LICLOC configuration type.

**Note:** COMMON-LICLOC configuration type cannot be updated.

## APAMA-ENVVAR

- To define environment variables for correlator instance :

```
cc update configuration data local Apama-correlator-myCorrelator
APAMA-ENVVAR -i Correlator.properties
```

- To define environment variables for IAF instance:

```
cc update configuration data local Apama-iaf-myIAF
APAMA-ENVVAR -i IAF.properties
```

Example format of defining environment variables in a properties file:

```
MY_HOME=C:\project1
MY_BIN=${env:MY_HOME}\bin
MY_CONFIG=${APAMA_WORK}\project\${INSTANCE_NAME}\configs
```

## COMMON-MEMORY

- To update the memory settings of a Display Server:

```
cc update configuration data local Apama-displayserver-myDisplayServer
COMMON-MEMORY --input C:\MemoryConfiguration.xml
```

- To update the memory settings of a Data Server:

```
cc update configuration data local Apama-dataserver-myDataServer
COMMON-MEMORY --input C:\MemoryConfiguration.xml
```

Example format of the XML file MemoryConfiguration.xml:

```
<MemorySettings>
  <InitSize>256</InitSize>
  <MaxSize>1024</MaxSize>
  <ExtendedProperties>
    <Property name="-XX:MaxPermSize">128M</Property>
    <Property name="-XX:MaxDirectMemorySize">1G</Property>
    <Property name="-DProperty1">Value1</Property>
  </ExtendedProperties>
</MemorySettings>
```

- To fetch the memory settings of a Display Server:

```
cc get configuration data local Apama-displayserver-myDisplayServer
COMMON-MEMORY
```

- To fetch the memory settings of a Data Server:

```
cc get configuration data local Apama-dataserver-myDataServer
COMMON-MEMORY
```

## COMMON-COMPONENT-ENDPOINTS

- To create the endpoints configuration for a Display Server instance:

```
cc create configuration data local Apama-displayServer-myDisplayServer
COMMON-COMPONENT-ENDPOINTS -i AddDisplayServerEndpoint.xml
```

- To update the endpoints configuration for a Display Server instance for type correlator:

```
cc update configuration data local Apama-displayServer-myDisplayServer
COMMON-COMPONENT-ENDPOINTS-correlator-alias1
-i UpdateDisplayServerEndpoint.xml
```

- To update the endpoints configuration for a Display Server instance for type dataserver:

```
cc update configuration data local Apama-displayServer-myDisplayServer
COMMON-COMPONENT-ENDPOINTS-dataserver-alias1
-i UpdateDisplayServerEndpoint.xml
```

- To delete the endpoints configuration for a Display Server instance for type correlator:

```
cc delete configuration data local Apama-displayServer-myDisplayServer
COMMON-COMPONENT-ENDPOINTS-correlator-alias1
```

- To delete the endpoints configuration for a Display Server instance for type dataserver:

```
cc delete configuration data local Apama-displayServer-myDisplayServer
COMMON-COMPONENT-ENDPOINTS-dataserver-alias1
```

- To create the endpoints configuration for a Data Server instance:

```
cc create configuration data local Apama-dataserver-myDataServer
COMMON-COMPONENT-ENDPOINTS -i AddDataServerEndpoint.xml
```

- To update the endpoints configuration for a Data Server instance for type correlator:

```
cc update configuration data local Apama-dataserver-myDataServer
COMMON-COMPONENT-ENDPOINTS-correlator-alias1
-i UpdateDataServerEndpoint.xml
```

- To update the endpoints configuration for a Data Server instance for type `dataserver`:

```
cc update configuration data local Apama-dataserver-myDataServer
COMMON-COMPONENT-ENDPOINTS-dataserver-alias1
-i UpdateDataServerEndpoint.xml
```

- To delete the endpoints configuration for a Data Server instance type `correlator`:

```
cc delete configuration data local Apama-dataserver-myDataServer
COMMON-COMPONENT-ENDPOINTS-correlator-alias1
```

- To delete the endpoints configuration for a Data Server instance type `dataserver`:

```
cc delete configuration data local Apama-dataserver-myDataServer
COMMON-COMPONENT-ENDPOINTS-dataserver-alias1
```

Example format of the XML file for correlator for the instance ID `COMMON-COMPONENT-ENDPOINTS-correlator-alias1`:

```
<Endpoint alias="alias1">
  <Transport>
    <Host>localhost</Host>
    <Port>15903</Port>
  </Transport>
  <Auth/>
  <ExtendedProperties>
    <Property name="isRaw">true</Property>
    <Property name="type">correlator</Property>
  </ExtendedProperties>
</Endpoint>
```

Example format of the XML file for Data Server for the instance ID `COMMON-COMPONENT-ENDPOINTS-dataserver-alias1`:

```
<Endpoint alias="alias1">
  <Transport>
    <Host>localhost</Host>
    <Port>2888</Port>
  </Transport>
  <Auth/>
  <ExtendedProperties>
    <Property name="type">dataserver</Property>
  </ExtendedProperties>
</Endpoint>
```

- Viewing log files. You can view the log files of Apama component instances in the Command Central web interface and command line interface.

- To list the log files of a correlator instance:

```
cc list diagnostics logs local Apama-correlator-myCorrelator
```

- To retrieve log entries from a log file of a correlator instance:

```
cc get diagnostics logs local Apama-correlator-myCorrelator
Correlator.err
```

- To export log files of a correlator instance:

```
cc get diagnostics logs local Apama-correlator-myCorrelator
Correlator.err+Correlator.out export -o CorrelatorLog.zip
```

- Monitoring Key Performance Indicators (KPIs) for correlator. You can view basic KPIs for each correlator instance that is online.

- To view KPIs in command line interface, use the command:

```
cc get monitoring runtimestate local Apama-correlator-myCorrelator
```

For information about all the properties and supported configuration types, see "Administering Apama" in *Software AG Command Central Help*.

For information about all the options for `cc` command, see "Options for the Commands" in *Software AG Command Central Help*.

For more information, see *Software AG Command Central Help*.

## Deploying an Apama project in Command Central

You can follow these steps to deploy a working Apama project in Command Central:

- To deploy a working Apama project in Command Central, you must first export the Apama project to a directory using the Ant Export wizard. For information on using the Ant Export wizard, see "Exporting to a deployment script" in *Using Apama with Software AG Designer*.

The exported Apama project directory contains configuration files that can be used to configure the Apama component instances created in Command Central.

- The SPM server must be able to access the configurations of adapters and correlator. You can copy the exported Apama project directory to the SPM server.

You can also share the exported Apama project directory with the SPM server, and map or mount the directory on SPM server so that the configuration files are accessible locally. For example, map the exported Apama project directory on Command Central as `H:\Apama_Config_Export`.

- Configure the Apama component instances in Command Central.
  - For the correlator, inject the monitors to the correlator instance created in Command Central using the `ant` command.

```
ant target_name -Dcorrelator.host=correlator_host  
-DdefaultCorrelator.port=correlator_port_value
```

For example, if you have exported your Apama project to `C:\Apama_Config_Export`:

```
C:\Apama_Config_Export>ant inject-correlator-Default Correlator  
-Dcorrelator.host=localhost -DdefaultCorrelator.port=15903
```

- For the IAF, create the IAF instance in Command Central using the IAF configuration file available in the exported directory. The IAF configuration file is located in the exported directory at `\adapters\iaf_config.xml`. For example, while creating an IAF instance in Command Central, provide the `iafConfigFile` property as:

```
iafConfigFile=H:\Apama_Config_Export\adapters\iaf_config.xml
```

- For the Dashboard servers, if your configuration needs to access any resources from the Apama project, you must map or mount the dashboards folder in the project directory on the SPM server.

## About deploying Apama applications with a YAML configuration file

As of 9.10.0.4, it is possible to deploy Apama applications with a YAML configuration file.

Instead of having another process inject code and send events into a correlator at startup, it is possible to use a YAML configuration file to list files to be loaded by the correlator at startup. This is useful for Docker containers or other minimal environments where only part of an Apama installation is available or it is not practical to run Java tools to perform the injections. It is also a better fit for Docker use cases as the correlator does not require any other coordination process for startup. For typical installations not using such environments, use of Ant macros or Command Central is recommended instead, which perform the injections after starting the correlator.

The YAML configuration file for the correlator is specified using the `--config` option when starting the event correlator (see also ["Starting the event correlator" on page 130](#)). The YAML file itself contains the following:

```
correlator:
  initialization:
    list:
      - ${PARENT_DIR}/bin/myPlugin.jar
      - ${PARENT_DIR}/eventdefinitions/evtdef.mon
      - ${APAMA_HOME}/monitors/ConnectivityPlugins.mon
      - ${PARENT_DIR}/monitors/app.mon
      - ${PARENT_DIR}/events/start.evt
    encoding: UTF8
```

It is recommended to use a substitution variable such as `${APAMA_HOME}` or `${PARENT_DIR}` rather than absolute or relative paths. This makes the configuration independent of the correlator's current working directory. `${PARENT_DIR}` is set to the directory containing the configuration file.

The `list` entries must have one of the following extensions:

- `.mon` for EPL monitor, aggregate and event definition source.
- `.jar` for JMon or EPL plug-ins written in Java.
- `.cdp` for correlator deployment packages.
- `.evt` for event files.

Note that Apama queries (`.qry`) or scenario definitions (`.sdf`) are not supported in source form. Scenarios have to be generated into EPL, and the generated EPL needs to be listed in the YAML configuration.

The `encoding` entry is optional. If `UTF8` is specified, all of the text input files (`.mon`, `.evt`) are read as UTF-8. If `local` is specified or if the `encoding` entry is not specified at all, the text files are assumed to be in the local encoding unless they start with a UTF-8 byte order mark (BOM) in which case they are treated as UTF-8.

This mechanism separates the build-time (calculating injection order, generating EPL) steps from the deployment-time steps, so no build steps are required in the environment where the correlator is running. This does mean that any changes to the project potentially require rewriting the YAML list and then redeploying, however, it allows separation of these concerns.

## About deploying components with EMM

---

Apama's Enterprise Management and Monitoring (EMM) console provides a graphical interface for configuring, deploying, and monitoring various Apama components across multiple hosts.

**Note:** Apama recommends you to use Command Central for all deployment tasks instead of EMM.

- For a general overview of EMM, see ["Using the Management and Monitoring Console" on page 27](#).
- For information on using EMM to start and manage correlators, see ["Deploying and Configuring Correlators" on page 58](#).
- For information on using EMM to start and manage adapters, see ["Deploying and Configuring Adapters" on page 79](#).

## About Apama command line utilities

---

Apama provides a variety of command line tools for managing and monitoring Apama correlators. For information and instructions on using these tools to monitor and manage event correlators, see ["Correlator Utilities Reference" on page 129](#).

## About deploying dashboards

---

Dashboard deployment and administration involves the following activities:

- Deployment package installation and configuration — see ["Deploying Dashboards" in \*Building and Using Dashboards\*](#).
- Data Server and Display Server management — see ["Managing the Dashboard Data Server and Display Server" in \*Building and Using Dashboards\*](#).



- Security administration — see "Administering Dashboard Security" in *Building and Using Dashboards*.

Before you perform these tasks, you should familiarize yourself with the deployment and administration concepts described in "Dashboard Deployment Concepts" in *Building and Using Dashboards*.

### Deployment options

Dashboards can be deployed as simple, thin-client Web pages, as Java applets, as Java Web Start applications, or as files that can be loaded into a locally-installed, desktop application, the Dashboard Viewer. These deployment options are described and compared in "Deployment options" in *Building and Using Dashboards*.

### Data server and display server

Scalability and security of dashboard deployment are supported by the use of the *Data Server* and *Display Server*, which mediate dashboard access to running scenarios and DataViews. The Data Server and Display Server are introduced in "Data Server and Display Server" in *Building and Using Dashboards*.

### Process architecture

Applet and Web Start dashboards communicate with the Data Server via servlets running on an application server. Simple, thin-client, Web-page dashboards communicate with the Display Server via servlets running on your application server. Locally-deployed dashboards communicate directly with the Data Server. The structure of deployed configurations is detailed in "Process architecture" in *Building and Using Dashboards*.

### Builders and administrators

Dashboard deployment involves the use of a dashboard deployment package that was generated by Apama's Dashboard Deployment Configuration Editor in Software AG Designer. In some cases the user that generated the deployment package is different from the user that installs and configures the deployment and administers the Data Server. The information that must be transferred between these two types of users is discussed in "Builders and administrators" in *Building and Using Dashboards*.

## About tuning applications for performance

---

The performance of Apama applications can be enhanced by employing multiple correlators. For information about strategies for deploying multiple correlators and the Apama tools available for accomplishing this, see ["Tuning Correlator Performance" on page 103](#). The section also contains information about preserving a correlator's runtime state.

## Setting up the environment using the Apama Command Prompt

---

Before you can run any of the Apama tools (such as `engine_send` or `engine_inject`) or any of the Apama servers (such as the correlator or the Integration Adapter Framework which is also known as the IAF) from a normal command prompt, you have to set up your environment correctly. This includes setting the paths to the Apama installation directory, the Apama work directory, the location of the libraries, the Java location, and other environment variables. Apama provides a batch file (Windows) or shell script (UNIX) for this purpose, which is called the "Apama Command Prompt".

- **On Windows**, you invoke the Apama Command Prompt by choosing the following command from the Start menu:

**Software AG > Tools > Apama *n.n* > Apama Command Prompt *n.n***

Keep in mind that "Software AG" is the default group name that you can change during the installation.

Alternatively, if you are already in a regular Windows command prompt, you can run the file `apama_env.bat` which is located in the `bin` directory of your Apama installation.

- **On UNIX**, you invoke the shell script from a Bash shell. Please note `csh` (C Shell) is not supported. Use the following command from within your Apama installation directory:

```
source bin/apama_env
```

It is important that you use `source` because invoking `apama_env` directly will not work.

You can add the above command to your shell initialization script (which is `.bashrc` in the case of the Bash shell). If you do so, every shell you use will be an Apama Command Prompt.

# 3

## Using the Management and Monitoring Console

---

■ Installation .....	28
■ Concepts .....	29
■ State persistence .....	32
■ Managing licenses .....	33
■ Managing security .....	34
■ Managing hosts .....	35
■ Managing components .....	45
■ Component status indicators .....	46
■ Working with components .....	47
■ Configuring components with the details pane .....	51
■ Preferences .....	52
■ Deploying and Configuring Correlators .....	58
■ Deploying and Configuring Adapters .....	79

**Note:** Apama recommends you to use Command Central for all deployment tasks instead of EMM.

This section describes how to use Apama's Enterprise Management and Monitoring (EMM) console to configure, deploy, and monitor Apama components.

EMM allows centralized, graphical management and monitoring of an Apama installation. From it, you can configure, start, stop and monitor Apama software components (correlators and IAF integration adapters) across multiple hosts. You can also centrally install and upgrade licenses.

## Installation

EMM can be run from any computer that has network access to the hosts where Apama components are installed. That is, it does not have to be installed on the machine on which the other Apama components are to be run, although doing so is not detrimental as long as the host is powerful enough to cope with the performance requirements of the application being run on Apama.

**Note:** In this release, EMM is supported on only Windows platforms. However, it can manage components running on all supported hosts. For a complete list of supported platforms, see the *Supported Platforms* document for the current Apama version. This is available from the following web page: <http://documentation.softwareag.com/apama/index.htm>.

Once installed on the host of choice, you can run EMM by selecting the **Software AG > Tools > Apama n.n > Apama Management and Monitoring n.n** menu item from the Windows **Start** menu.

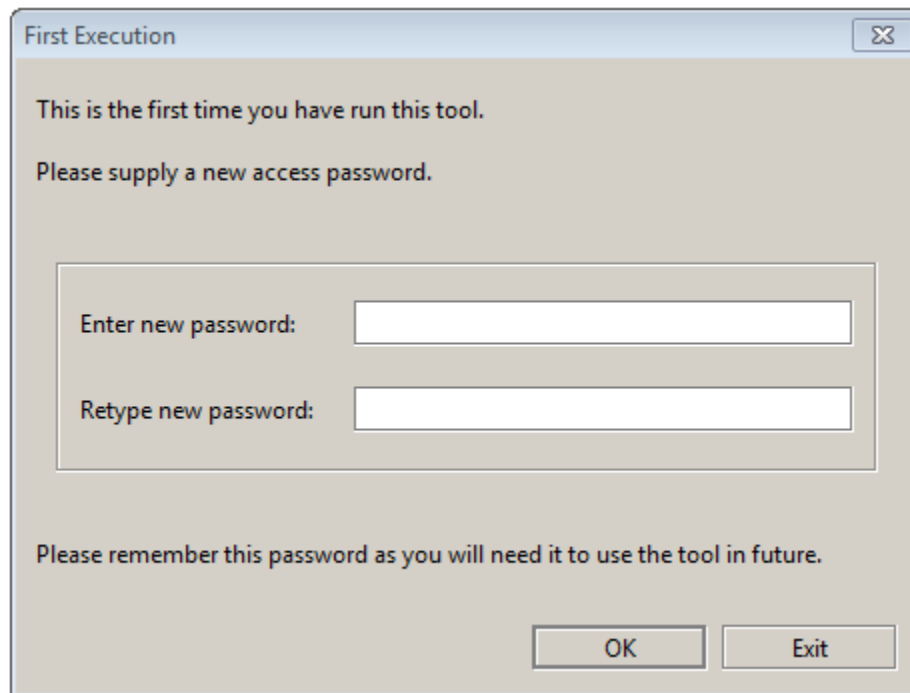
Alternatively, you can run it from the Apama command prompt as follows (see also "[Setting up the environment using the Apama Command Prompt](#)" on page 26):

```
emm
```

When EMM is run for the first time you will be asked to configure an access password, as shown in the illustration, below.

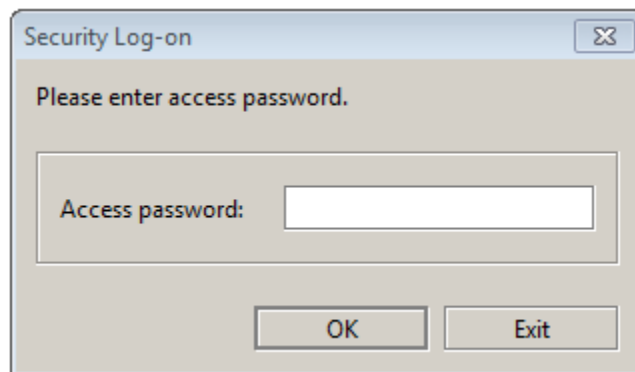
- If you enter a password here, the password will be required whenever EMM is launched. It can be changed subsequently through the **Options** menu.
- If you do not enter a password here (i.e., leave the field blank), EMM will not prompt for a password when it starts up. This can be useful if you want to start EMM from a script.

**Note:** It is important that you keep a record of the access password, as you will be asked to enter it on all subsequent uses of the tool. If you forget the password you will have to reinstall the tool from the original installation distribution.



A dialog box titled "First Execution" with a close button (X) in the top right corner. The text inside reads: "This is the first time you have run this tool." followed by "Please supply a new access password." Below this, there are two text input fields. The first is labeled "Enter new password:" and the second is labeled "Retype new password:". At the bottom, there is a reminder: "Please remember this password as you will need it to use the tool in future." and two buttons labeled "OK" and "Exit".

If you are not running the tool for the first time, you will be asked to enter the access password:



A dialog box titled "Security Log-on" with a close button (X) in the top right corner. The text inside reads: "Please enter access password." Below this, there is a single text input field labeled "Access password:". At the bottom, there are two buttons labeled "OK" and "Exit".

Once you have entered the correct password, you then have access to all the capabilities of the tool.

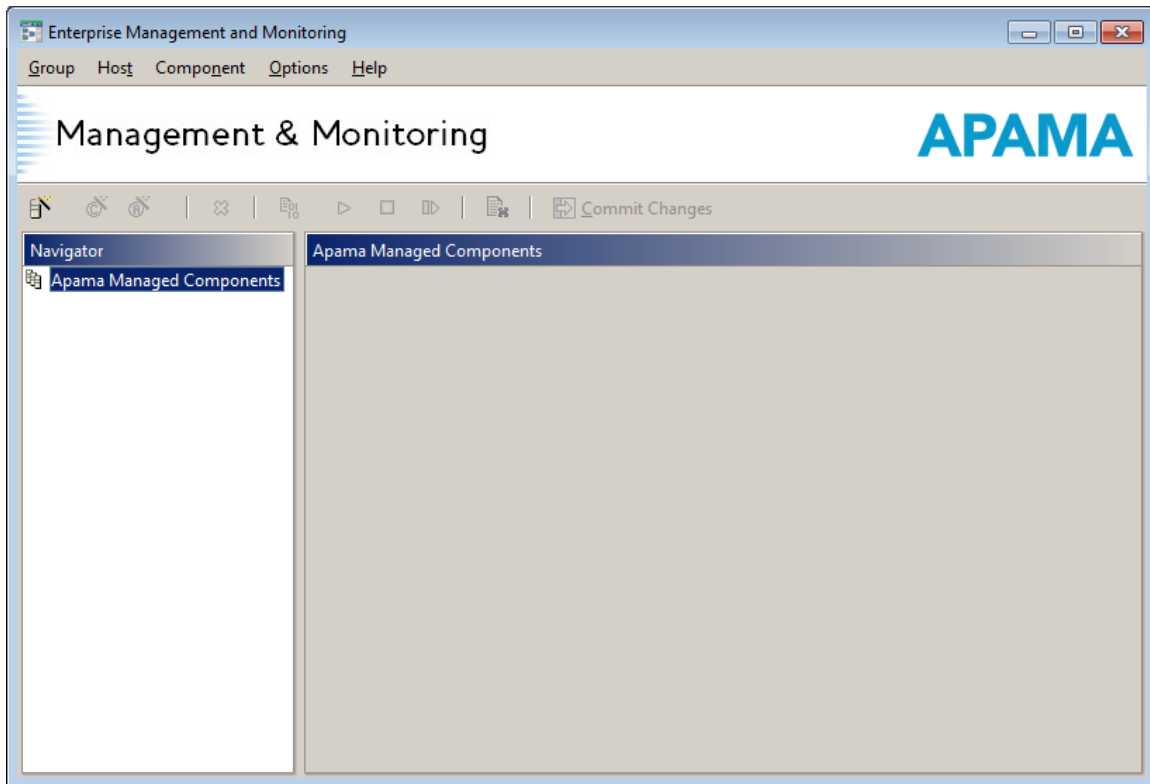
## Concepts

Two basic entities can be configured and manipulated in EMM: *hosts* and *components*. *Hosts* refer to personal computers, workstations or servers, running Solaris, Linux or Windows, which have Apama installations. In addition these machines must have had the *Sentinel Agent* (see "[Sentinel Agents](#)" on page 35) installed on them and it must be running. "[Managing hosts](#)" on page 35 describes the facilities provided by EMM for dealing with hosts.

*Components* refer to Apama event correlators and IAF adapters. Apama components can be run and manipulated by EMM on any host that is running the Sentinel Agent. ["Managing components" on page 45](#) describes the facilities provided by EMM for dealing with components.

## The EMM console window

The following illustration shows the key features of EMM's main window.



### Menubar

The Menubar is the main way of interacting with the tool and carrying out operations on hosts and components.

### Toolbar

The Toolbar contains a set of icons that provide a convenient graphical shortcut to the most common operations that can be carried out on hosts and components. These operations can also all be carried out from the Menubar.

### Navigation Pane

The Navigation Pane lists the set of hosts that EMM is aware of and the Apama components that are running on them. It also provides visual cues as to the current operating status of the hosts and components. Right clicking on an entity displays a

context-sensitive menu that lists the operations that can be carried out on that entity. These operations can also all be carried out from the Menubar and from the Toolbar.


### Details Pane

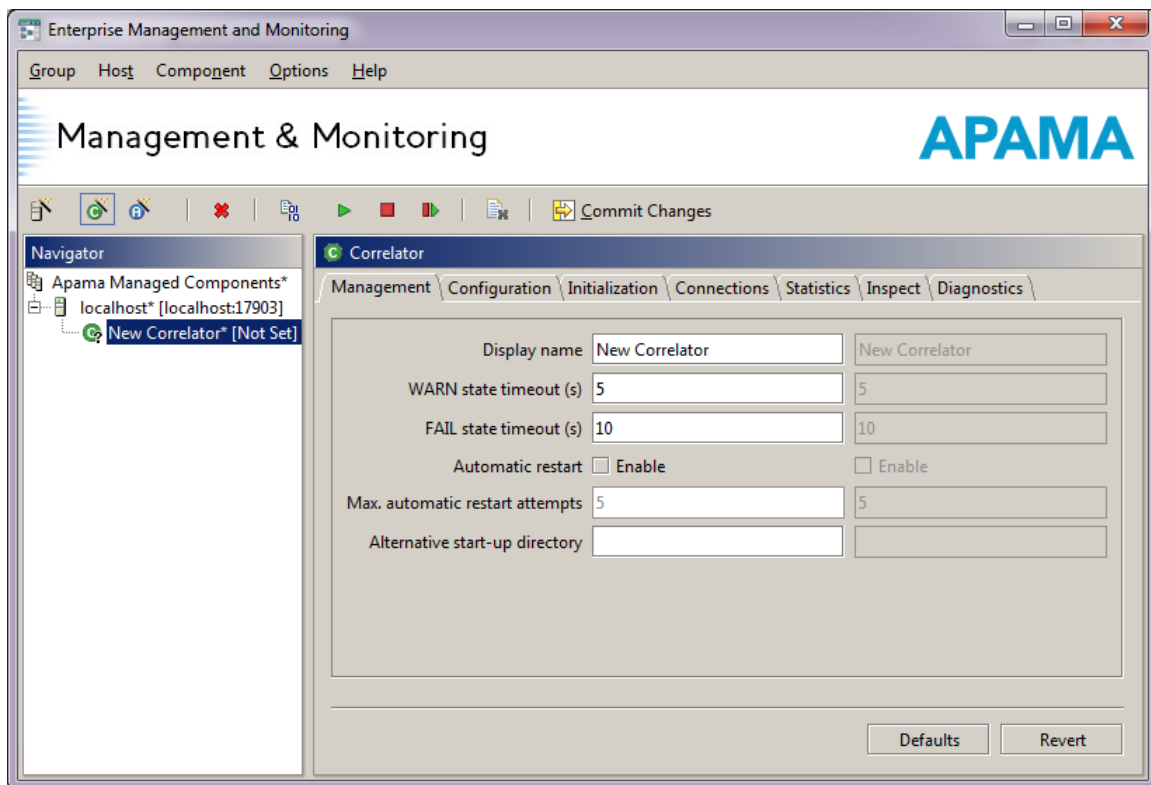
The Details Pane displays information that is relevant to the currently selected host or component. Its contents change accordingly, and can be used to both configure as well as to monitor and inspect the status of the selected item.

## Committed values and outstanding changes

EMM can configure and manipulate hosts and Apama components, and to do this it needs to maintain extensive configuration information.

Throughout EMM there are concepts of the *current*, *committed* configuration data and of *outstanding changes*, or *working copy* data.

On each Details Pane for a host or a specific component where you can supply configuration data, you will see two columns of values, as shown in the following illustration. The rightmost, grayed out values indicate the actual *committed* data – the values currently in use. The leftmost column is where you can provide the new values you would like to use. As long as they are just shown in the leftmost columns, these *outstanding changes* are just that – temporary working data. A component or host with unsaved changes will have a "\*" suffix in its entry in the Navigation Pane. For changes to take effect you need to commit them, using the **Commit Changes** button (  ) on the Menubar, or the relevant menu items on the **Group**, **Host** and **Component** menus. Note that some component options cannot be changed once the component has started, and these will not take effect until the component is restarted, even after being committed.



The **Commit Changes** action scope varies according to what is selected when it is pressed:

- if a component is selected in the Navigation Pane, then all outstanding changes in that component, and only in that component, are committed (the **Commit changes** item on the **Component** menu is enabled in this case),
- if a host is selected in the Navigation Pane, then all outstanding changes in that host, and in all its components, are committed (the **Commit changes** item on the **Host** menu is enabled in this case),
- if **Apama Managed Components** is selected in the Navigation Pane, then all outstanding changes throughout all of EMM's hosts and components are committed (the **Commit changes** item on the **Group** menu is enabled in this case).

See ["Managing hosts" on page 35](#) and ["Working with hosts" on page 38](#) for information on the **Details** Pane for hosts and for the specific components supported by the EMM console.

## State persistence

All of EMM's management state, including the list of hosts and components it is managing and the committed settings for each (including the diagnostic messages), are persisted by EMM to disk.



When EMM is stopped and restarted no settings are lost, although note that in the current version EMM's status monitoring and auto-restart capabilities are only active while the console is running.

The state is stored in a file in the Apama Work directory called `etc\emm_state.dat` and a backup is created in `etc\emm_state.dat.bak`. EMM must have the necessary permissions to read/write these files. If there is a problem locating or reading the state file EMM will offer to load from the backup file instead. If neither file can be located or read, EMM will revert back to its default installation stage configuration. Should EMM fail to save its state file correctly for whatever reason then temporary files of the form `serialXXXXX.tmp` will be created in the `etc` directory; these files can be safely deleted.

## Managing licenses

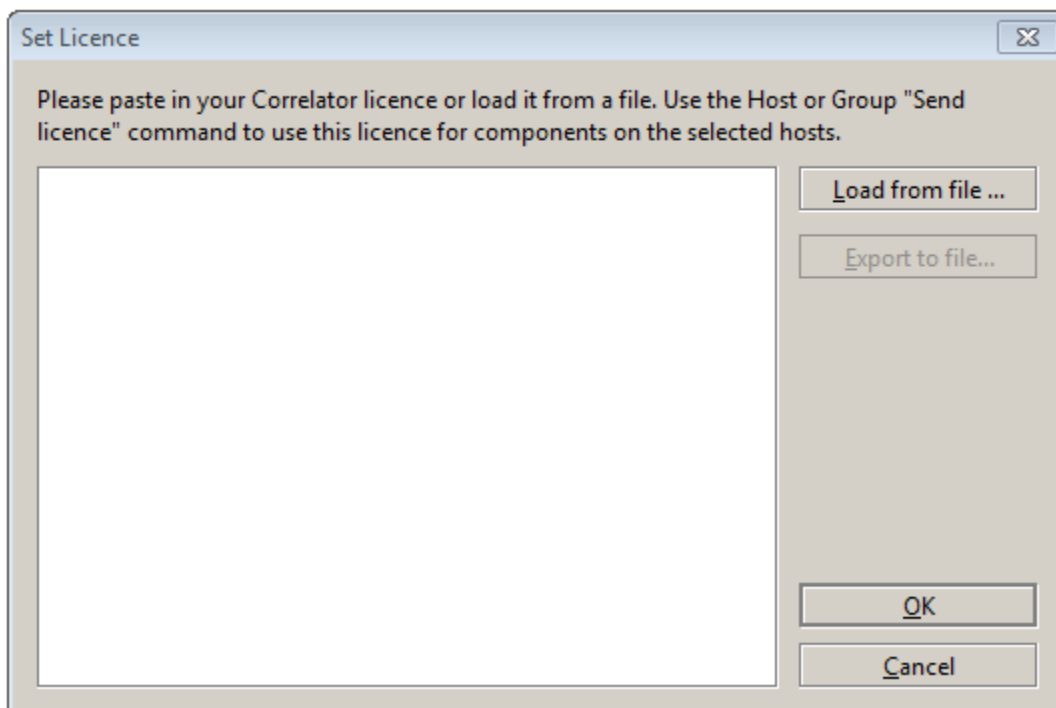
An Apama event correlator requires a valid Apama license. As well as allowing the correlator to start, a license specifies restrictions in its functionality and performance. A file containing a valid license must be available to each correlator all the time.

EMM aids license deployment and maintenance by its facility to push out licenses to any host where Apama correlators are to be started (or are already running). This facility also allows updating of existing installed licenses.

### To load a license for use with EMM

1. Select **Set License...** from the **Options** menu option on the menu bar.

This displays the **Set License** dialog box shown in the following illustration.



In theory you can type in the license but as it is quite a long string and must be entered exactly as it was sent to you by Apama Technical Support, it is greatly preferable to read it in from a text file.

2. Select **Load from file ...**, locate the license file from where you saved it, and **Open** it.
3. Click **OK** to set this as the current active license to be used by EMM.

However, note that this license is only actually sent out to hosts when the **Send license** option is applied to the host entity. Alternatively, the licenses on all known hosts can be updated at the same time by selecting **Apama Managed Components** in the Navigation Pane and choosing the **Send License** option from the **Group** menu.

**Note:** EMM will not attempt to validate your license; therefore it is important that you enter it correctly. If you attempt to start a correlator and the host has no license file, the correlator will run with reduced capabilities (see "Running Apama without a license file" in *Introduction to Apama*). It will be flagged with a warning icon to indicate that it has started in this state (look at the component's **Diagnostics** tab for more information).

---

## Managing security

---

When you run EMM, if you have configured a password as described in "[Installation](#)" on page 28, the start up process will always request an access password. If you configured a password when you first ran EMM, this step cannot be bypassed. Should you forget the access password, you will need to reinstall the tool from the Apama installation media.

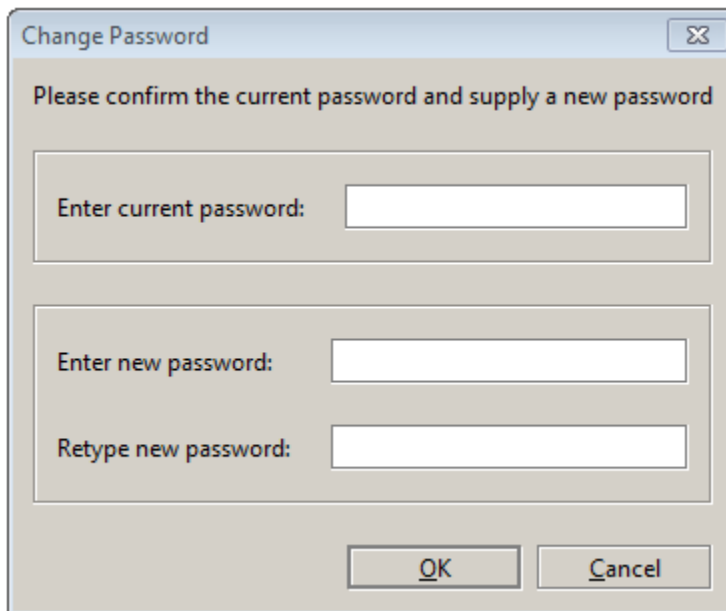
If you left the password field empty as described in "[Installation](#)" on page 28, you are not prompted to enter a password.

---

### To change the password

1. Select **Change Password...** from the **Options** menu option on the Menubar.

This displays the Change Password dialog, shown in the following illustration.



A screenshot of a 'Change Password' dialog box. The dialog has a title bar with the text 'Change Password' and a close button (X). Below the title bar, it says 'Please confirm the current password and supply a new password'. There are three input fields: 'Enter current password:', 'Enter new password:', and 'Retype new password:'. At the bottom, there are 'OK' and 'Cancel' buttons.

## Managing hosts

EMM can start and stop components on any supported platform within your network, as long as that host has had the relevant Apama components installed on it, and as long as it is running a Sentinel Agent.

## Sentinel Agents

A Sentinel Agent is a small process that can start and stop Apama correlators and IAF adapters on the host where it is running. EMM starts and stops components on a particular host in a platform independent way by interacting with the Sentinel Agent running on that host. Therefore, a Sentinel Agent must be running on any machine where you intend to manipulate Apama components using EMM.

On Windows and UNIX, the Sentinel Agent can be started in one of these ways:

- **By starting the service.** To do so, you must have administrator (Windows) or root (UNIX) privileges.

Before you can start the service, you first have to add it with the following command:

```
apama_services -add -s sentinel
```

After the service has been added, you can start it with the following command:

```
apama_services -start -s sentinel
```

You can stop the service with the following command:

```
apama_services -stop -s sentinel
```

You can uninstall the service with the following command:

```
apama_services -delete -s sentinel
```

- **By starting the executable.** If you do not want to use the service, you can also start the Sentinel Agent executable manually. See ["Using the sentinel\\_agent tool" on page 36](#) for further information.

**Important:** If you install the Sentinel Agent as a service or run it by starting the executable, you are responsible for stopping and/or uninstalling it before running the Software AG Uninstaller.

Any information written to `stdout` or `stderr` by components started with a Sentinel Agent will be logged in the Agent's log directory in files called `<component-type>-<PID>-std{out,err}.log` on UNIX and `<component-type>-<num>-std{out,err}.log` on Windows, where `<PID>` is the process ID and `<num>` is an integer number. Valid component types are `correlator` and `adapter`. The `stdout` / `stderr` logs will be deleted by Sentinel when the relevant component is stopped if they are empty (which should normally be the case).

## Using the sentinel\_agent tool

The executable for this tool is located in the `bin` directory of the Apama installation. Running the tool in the Apama Command Prompt (see ["Setting up the environment using the Apama Command Prompt" on page 26](#)) ensures that the environment variables are set correctly.

### Synopsis

To start the Sentinel Agent, run the following command:

```
sentinel_agent [ options ]
```

When you run this command with the `-h` option, the usage message for this command is shown.

### Description

The license is assumed to be in `APAMA_WORK\license\ApamaServerLicense.xml` unless overridden on the command line. The Apama work directory is set from the `-w` argument or from the `APAMA_WORK` environment variable if `-w` is not specified.

If the license is not found at the above location, the correlator will also search `Apama/etc/ApamaServerLicense.xml` in the installation directory.

### Options

The `sentinel_agent` tool takes the following command line options:

Option	Description
<code>-V   --version</code>	Prints out the Sentinel Agent's version information. Does not start the Sentinel Agent.
<code>-h   --help</code>	Displays usage information. Does not start the Sentinel Agent.
<code>-p port   --port port</code>	Name of the host on which the Sentinel Agent will listen for connections from EMM. The default is 17903.
<code>-f file   --logfile file</code>	The file where to output logging information from the Sentinel Agent. If a log file is not specified and the Sentinel Agent was run as a normal executable from a Command Prompt or a Terminal window, it will log to the system configured <code>stderr</code> . If the Sentinel Agent was run as a Windows service, a log file must be specified; otherwise the Agent will not start.
<code>-v level   --loglevel level</code>	Sets the logging level. The log levels possible, in increasing order, are <code>TRACE</code> , <code>DEBUG</code> , <code>INFO</code> , <code>WARN</code> , <code>ERROR</code> , <code>CRIT</code> , and <code>FATAL</code> , where each level includes all subsequent ones. For example, <code>WARN</code> includes all <code>WARN</code> level, all <code>ERROR</code> level, all <code>CRIT</code> level, and all <code>FATAL</code> level log messages. If not specified, the default log level is <code>WARN</code> .
<code>-t   --truncate</code>	If set, the log file will be emptied at start up, otherwise it will be appended to.
<code>-N name   --name name</code>	Sets the name of the component to <i>name</i> .
<code>-l file   --license file</code>	A reference to an Apama license file. If provided, the Sentinel Agent will be able to pass this license file's location on to the Apama components that it starts. Alternatively, a license file can be sent to the Sentinel Agent dynamically via EMM.
<code>-H dir   --apamahome dir</code>	The root directory where the Apama executable files are installed, as selected during installation.

Option	Description
<code>-W dir   --apamawork dir</code>	The root directory where the Apama work files are located, as selected during installation.
<code>-Xconfig file   --configFile file</code>	Uses the specified service configuration file. For more information about this configuration file, see <a href="#">"Using the Apama component extended configuration file" on page 225</a> .

## Working directory

The Sentinel Agent's current working directory/folder becomes the current working directory of all components started by the Agent, unless an **Alternative start-up directory** was configured for the component. The working directory is important because most of the files and paths specified when configuring components in EMM are assumed to be relative to the Sentinel Agent's working directory if a full absolute path is not provided.

### Determining the working directory

If the Sentinel Agent was started manually then its current working directory is simply the current directory when it was run.

If it was started automatically – either as a service on Windows or as a daemon on UNIX – then the current working directory is determined as follows.

On Windows:

1. Normally, the Sentinel Agent tries to use the `logs` subdirectory of the Apama work directory.
2. Otherwise the Sentinel Agent tries to use the `%WinDir%\System32` directory.
3. If the Sentinel Agent cannot write to any of these directories, it will fail to start.

On UNIX:


1. The Sentinel Agent tries to use the `/logs` subdirectory of the Apama work directory.
2. If the Sentinel Agent cannot write to this directory, it will fail to start.

## Working with hosts

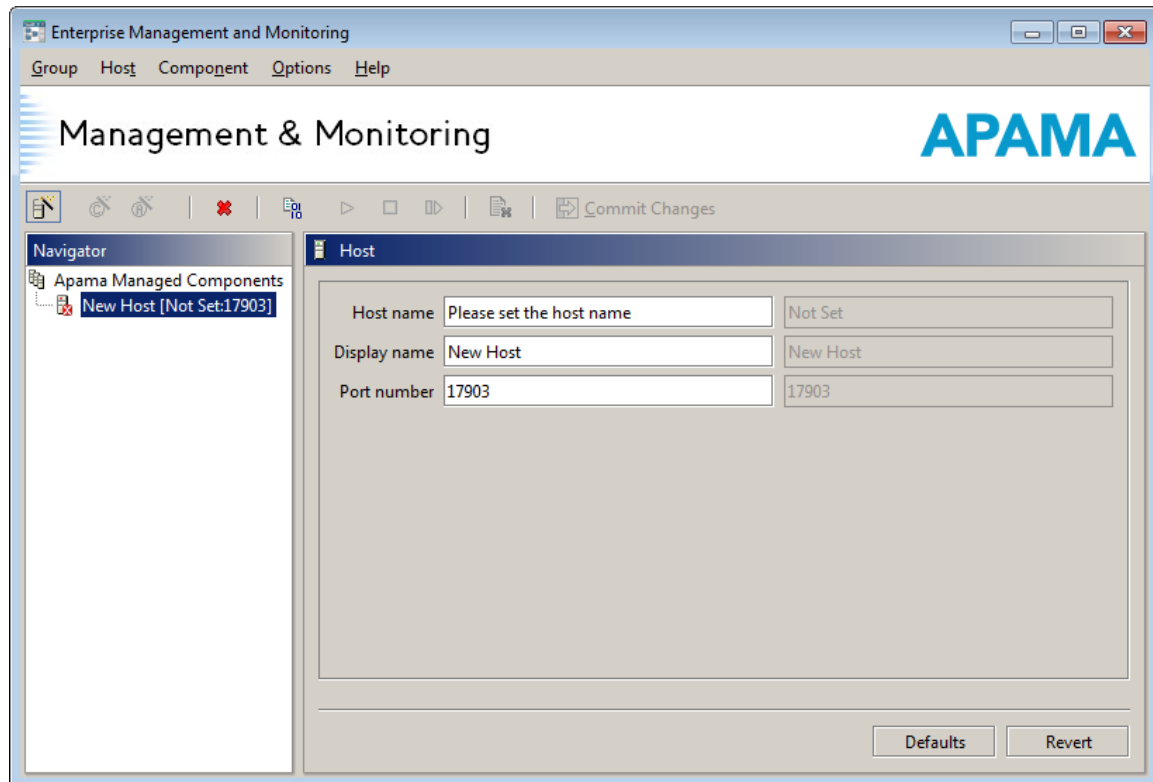
This documentation refers to the items on the EMM menu as the primary means of carrying out operations on hosts. However, these operations can also be carried out from the context menus in the Navigation Pane, and the most useful are also available on the toolbar.

The **Host** menu provides the options described below for interacting with hosts.

## Add host

This is equivalent to using the  button on the toolbar.

**Add host** allows you to add a host to the **Apama Managed Components** group, the root node illustrated in the Navigation Pane. This adds a placeholder node in the Navigation Pane and provides options for configuring EMM to work with the host in the Details Pane, as illustrated in the following.



You need to provide the following information:

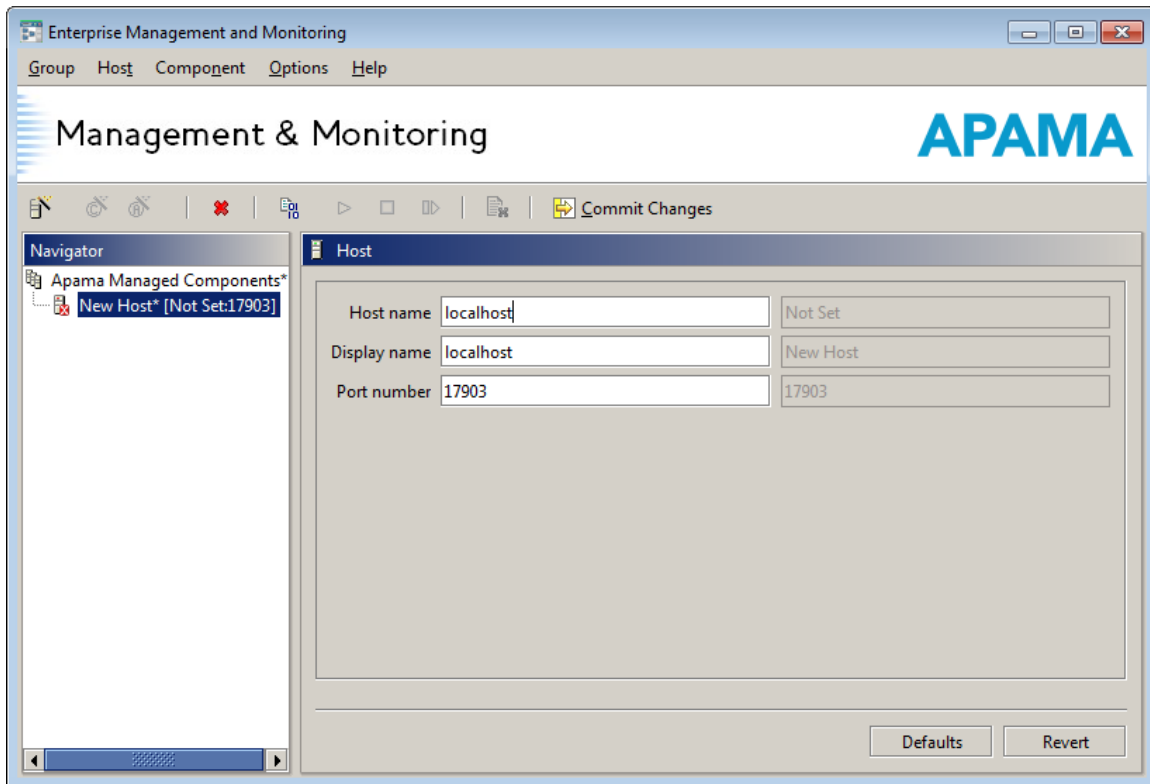
- **Display name** – A descriptive name for the host. This name is what is displayed in the Navigation Pane.
- **Host name** – The host's name or IP address, for example `lawrence.apama.com` or `127.0.0.1`. Whether you need to provide the full name as in this example, or whether an abbreviation (e.g. `lawrence`) will do depends on your network configuration and DNS/WINS settings, but in most circumstances it is advisable to use the host's full network name.

You can only use ASCII characters in a host name.

Note that it is not possible to change the **Host name** once components have been added to the host.

- **Port number** – The port that the Sentinel Agent (on that host) is listening on. By default this is `17903`, although it might have been changed by whoever installed and configured the Sentinel Agent on that host.

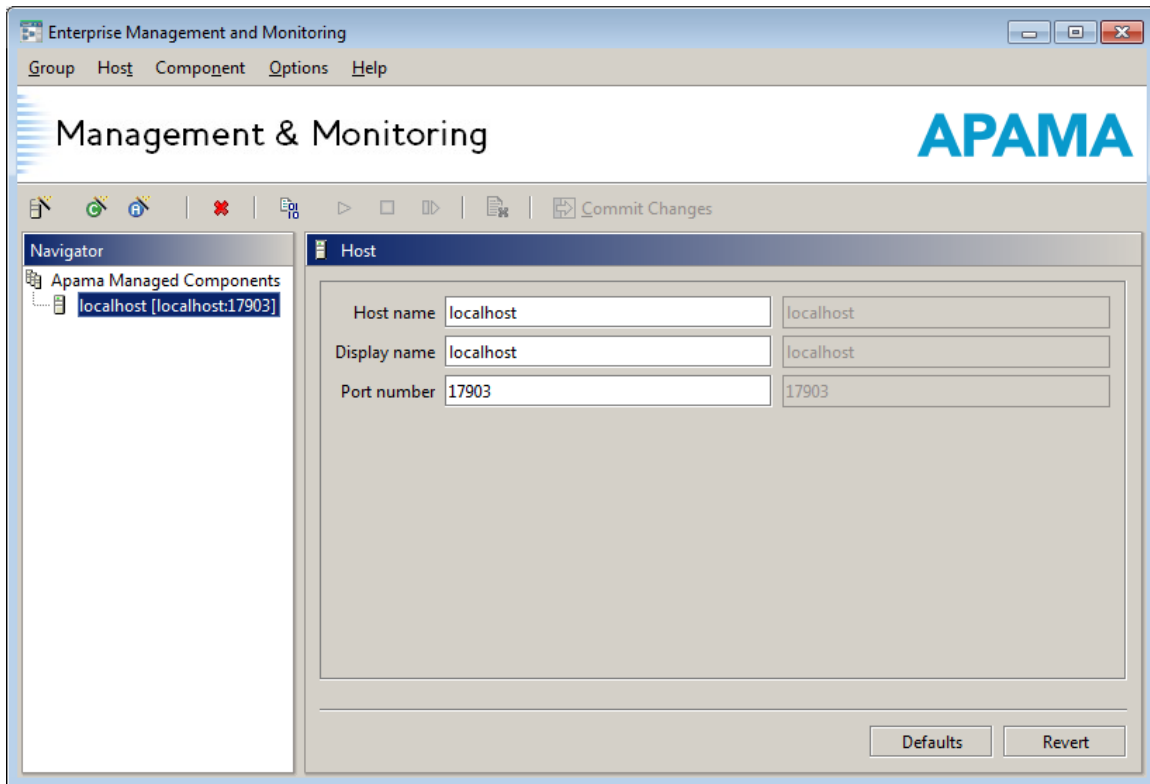
In following illustration, we have created a reference to the host `localhost` and specified port `17903` (the default port) as the port to use for communicating with the Sentinel Agent running on this machine.





Once the host information has been set, click on the **Commit Changes** button (  ) on the toolbar. Note that no components can be added to a host until it is committed.


The **Commit Changes** button will be disabled if the **Host name** has not yet been set or if the host has already been setup and committed, and there are no outstanding changes to commit on the host or any associated component.






Notice how the icons next to the new node in the Navigation Pane are different in the illustrations above. When you first define a new host the icon alongside it in the Navigation Pane will be . This will change to  once EMM successfully manages to contact the Sentinel Agent running on that host.

EMM will continue verifying communication with the Sentinel Agent on that host at regular intervals, for as long as that host is included in the **Apama Managed Components** group.

If EMM cannot communicate with a host's Sentinel Agent, or connects and subsequently loses connection, the icon against that host will change to  and stay that way until the problem is resolved. This might be the case if the host is no longer available on the network, the network has failed, the remote Sentinel Agent has been shut down or if a firewall is blocking the port that the EMM console uses to communicate with the Sentinel Agent.

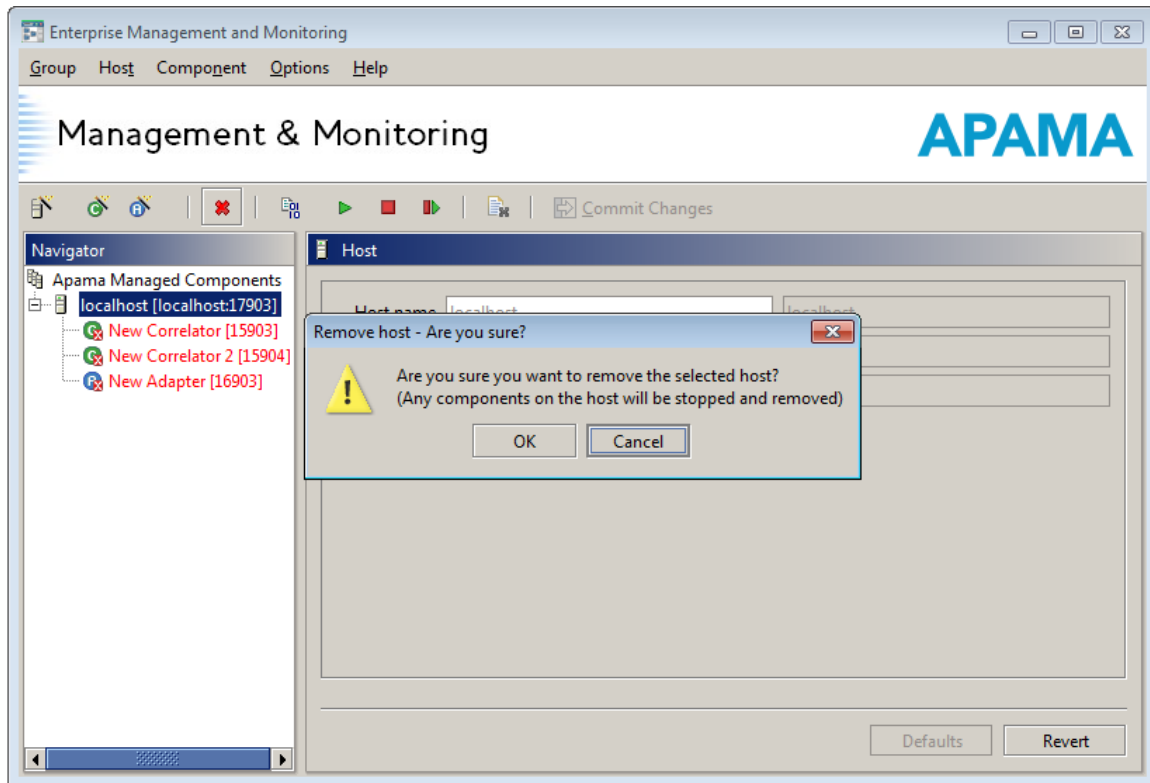
### Remove host

This is equivalent to using the  button on the toolbar, and is only available if a host is currently selected in the Navigation Pane.

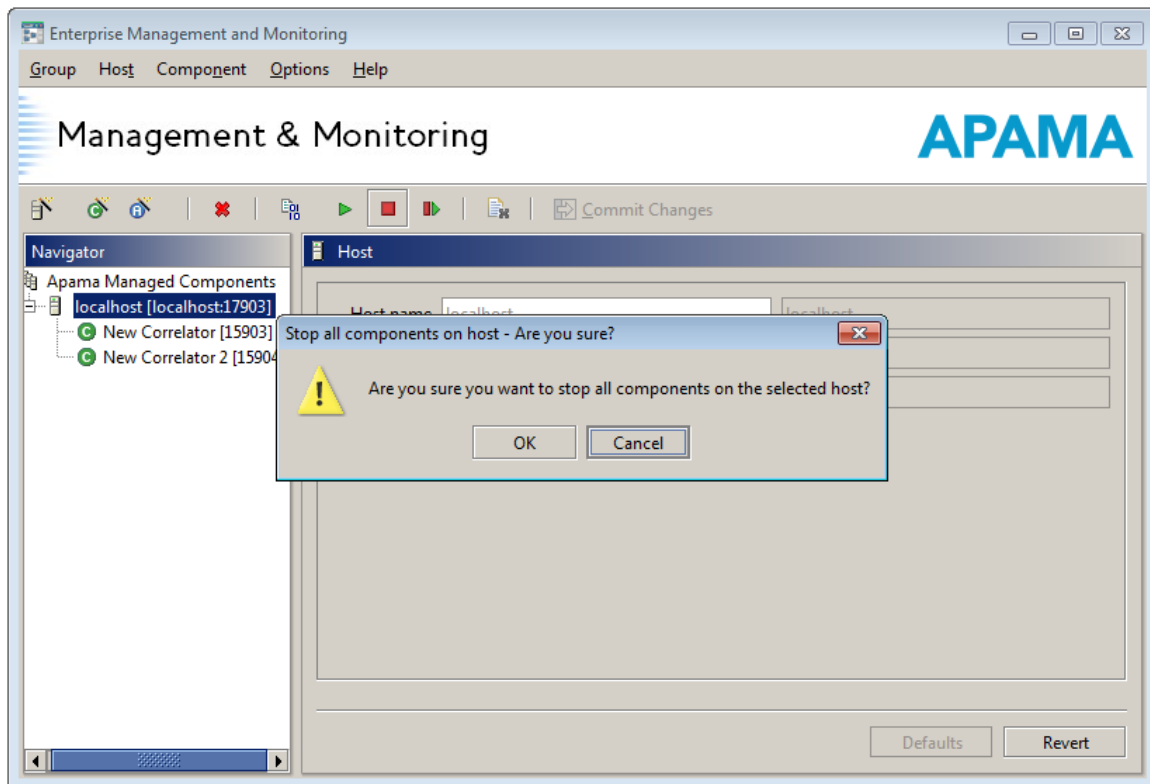
Choose it to remove all components on that host from the **Apama Managed Components** group.

Note that this operation does not shut down the remote Sentinel Agent on the host in question.

By default, EMM displays a confirmation dialog providing the option to cancel the removal of the host, as shown in the following illustration.




If the host contains one or more components that are known to be running, as well as providing the option to cancel the removal the dialog asks whether these component(s) should be stopped before being removed; see the following illustration.



Note: See ["Preferences" on page 52](#) for information about how these confirmation dialogs can be turned on or off using the EMM Preferences dialog.


### Start component(s)

This is equivalent to using the  button on the toolbar. This option is available if a host is currently selected in the Navigation Pane.

Choose it to start all Apama components configured on that host. See Chapter 3 for details on how to add and configure Apama components.


See ["Start component" on page 49](#) for information on why components might fail to start.

### Stop component(s)

This is equivalent to using the  button on the toolbar. This option is available if a host is currently selected in the Navigation Pane.


Choose it to stop all Apama components configured on that host.

### Restart component(s)

This is equivalent to using the  button on the toolbar. This option is available if a host is currently selected in the Navigation Pane.

Choose it to re-start (stop and start again) all Apama components configured on that host.

### Send license

This is equivalent to using the  button on the toolbar. This option is available if a host is currently selected in the Navigation Pane and a license has been configured within EMM (through the **Options / Set license** menu option).

Choose it to push out the currently configured license to the Sentinel Agent running on that host, which the Sentinel Agent will use to create a local license file on that host. The license file will be written to the location that was specified in the Sentinel Agent's command line startup parameters, replacing any existing license file.

A dialog will be displayed indicating if the license push was successful or whether the license was rejected as invalid or some other error occurred.

Components do not have to be restarted to become aware of the new license file.


## Managing all known hosts

All hosts defined within EMM are attached to the **Apama Managed Components** group item in the Navigation Pane.


The **Group** menu option allows one to run operations across all hosts in this group; that is all known hosts, without having to select them individually.

The operations available on the **Group** menu are also available by selecting the **Apama Managed Components** item in the Navigation Pane and then right-clicking to get its context-sensitive popup menu. While the item is selected you can also use the toolbar buttons to the same effect.

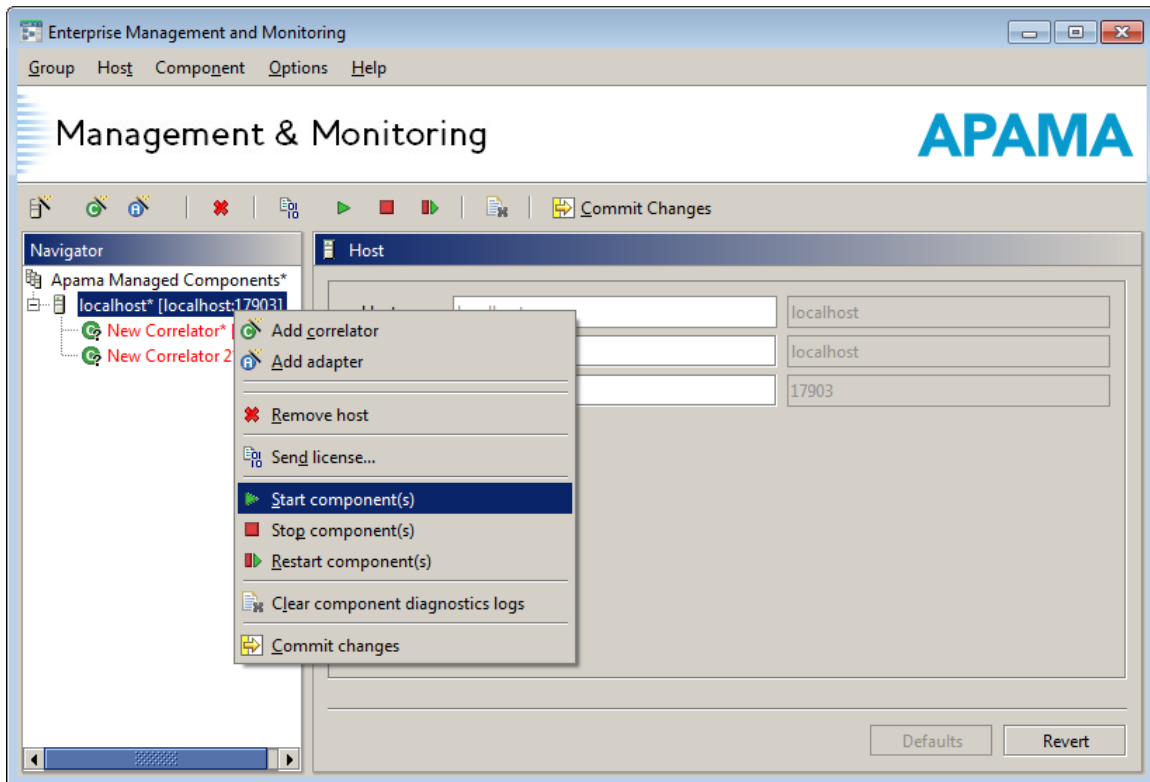
### Restart components

Choose this option to restart all components on all hosts. This is equivalent to clicking  on the toolbar.

### Start components

Choose this option to start all components on all hosts. This is equivalent to clicking  on the Menubar.


See ["Start component" on page 49](#) for information on why components might fail to start.



### Stop components

Choose this option to stop all components on all hosts. This is equivalent to clicking  on the Menubar.

### Send license

Update the license on all hosts. This is equivalent to clicking  on the Menubar.

See ["Send license" on page 44](#) for more information about license sending.

## Managing components

EMM can support various Apama components that can be used together to create a distributed Apama deployment.

The components are:

- Event Correlators – the event correlator is the core Apama event matching and analytic engine. See ["Add correlator" on page 47](#) and ["The correlator tabs" on page 60](#).
- IAF Adapters – the Integration Adapter Framework (IAF) allows rapid generation of integration adapters. These allow Apama to interface with an external source of

events like a message bus, an event feed, or a database. See ["Add adapter" on page 47](#) and ["The Adapter tabs" on page 81](#).



For information on how to use Apama command line utilities to run and configure the event correlator, see ["Correlator Utilities Reference" on page 129](#).






For information about using the adapters provided with Apama or creating adapters of your own, see:

- "Using Standard Adapters" in *Connecting Apama Applications to External Components*
- "Developing Custom Adapters" in *Connecting Apama Applications to External Components*


## Component status indicators

EMM can configure, start and stop components. For all the components it manages, EMM can monitor execution and automatically restart in case of failure (re-initializing where applicable).

As components are managed, their icons in the Navigation Pane will change to indicate their status. Although the base icons for a correlator and an IAF adapter are different, being  and , they have the same set of symbols overlaid on them to indicate their state. Using the correlator's icons as an example, the indicated states are:

-  - **UNKNOWN**. The state is still unknown, i.e. EMM is trying to communicate with the component.
-  - **STOPPED**. The component is not running and it is not supposed to be, that is, it has not been started, or it was explicitly stopped, in EMM.
-  - **RUNNING**. The component is running normally and it is supposed to be, that is, it was started from EMM.
-  - **STOPPING, STARTING**. A stop or start operation is in progress.
-  - **WARN**. This can mean one of the following:
  - The component is supposed to be running but is **not responding** – EMM has lost contact with it and is trying to reestablish communication. If this does not succeed the component will progress to the **FAIL** state.
  - The component itself started correctly but **one or more initialization actions failed**. Use the component's **Diagnostics** tab to investigate where the problem occurred. The warning can be cleared by pressing the **Clear warning** button on the **Diagnostics** tab.
  - The component has been started but it is **not supposed to be running** (e.g. it was started from outside EMM). The warning can be cleared by pressing the **Clear warning** button on the **Diagnostics** tab.
  - A correlator started on the local machine but there was no license file, so the correlator will run with reduced capabilities (see "Running Apama without a

license file" in *Introduction to Apama*). The warning can be cleared by pressing the **Clear warning** button on the **Diagnostics** tab, however Apama recommends supplying a valid license and restarting the component.

- The component itself started correctly but one or more **upstream connections to other components could not be established**. Use the component's **Diagnostics** tab to investigate where the problem occurred. The warning can be cleared by pressing the **Clear warning** button on the **Diagnostics** tab.
-  - **FAIL**. The component has failed. The component stopped responding to EMM for the entire WARN timeout configured for the it plus the FAIL timeout, so EMM now assumes that the component is no longer running.

In case of a FAIL check the **Diagnostics** tab to discover which of these situations occurred.

**Note:** The time to wait after a component stops responding before entering the **WARN** state, and the time between entering the **WARN** state and progressing to **FAIL** may both be configured on a per-component basis using the **Management** tab in the Details Pane.


## Working with components

This section refers to the items on the EMM menu as the primary means of carrying out operations on Apama components. However, these operations can also be carried out from the context menus in the Navigation Pane, and the most useful are also available on the toolbar.

Before operations can be carried out on these components, a Sentinel Agent must be installed and running on the target host, and the EMM console must have been configured to manage that host, as detailed in ["Managing hosts" on page 35](#).


The options described in the topics below are available from the **Components** menu when a host is selected.

### Add correlator

This option is only available if a host (or an existing component on a host) is selected in the Navigation Pane. It is equivalent to pressing the  button on the toolbar.


Choose this option to define and configure a new event correlator component on the selected host.

### Add adapter

This option is only available if a host (or an existing component on a host) is selected in the Navigation Pane. It is equivalent to pressing the  button on the toolbar.

Choose this option to define and configure a new IAF Adapter component on the selected host. This will add an Adapter to the host in the Navigation Pane, select it, and display the **Management** tab in the Details Pane.

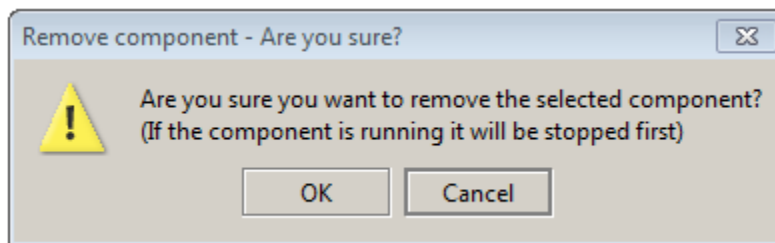
## Remove component

This option is only available if a component is selected in the Navigation Pane. It is equivalent to pressing the  button on the Menubar.

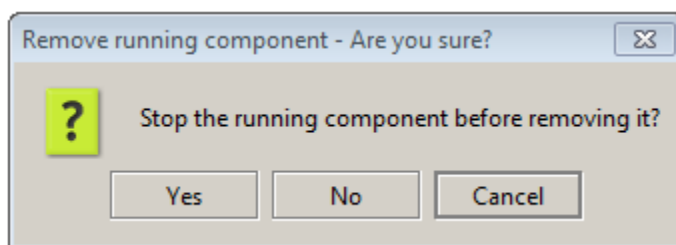
Choose this to remove the currently selected component from the Navigation Pane and stop it from being managed by EMM.

You can also remove a component by selecting it and pressing the **Delete** key.

By default, EMM displays a confirmation dialog providing the option to cancel the removal of the component, as shown below.



If the component is known to be running, as well as providing the option to cancel the removal the dialog asks whether the component should be stopped before being removed, as shown below.



**Note:** ["Warnings" on page 52](#) describes how these confirmation dialogs can be turned on or off using the EMM Preferences dialog.


## Move component to host

This option is only available if a component is selected in the Navigation Pane. When a component is selected and you select **Move component to host** from the **Component** menu, the Select New Host for Component dialog is displayed. Select the new host and click **OK**.





Any component type may be moved with this command. The moved component will have exactly the same configuration as it did on its previous host, including the same logical ID.

## Start component

This option is only available if a component is selected in the Navigation Pane. It is equivalent to pressing the  button on the toolbar.

Choose this option to start the currently selected component. The component's icon will change to indicate that the component is starting, and then to show it is operational. Note that this can take a few seconds since the icon is only updated once EMM can satisfactorily communicate with the component.

A component can fail to start for several reasons. The most common are:


- A valid license was not available on that host.
- The Sentinel Agent is not running on the host (check that its icon in the Navigator Pane is  and not )
- The component's executable was corrupt, or inaccessible due to user permissions issues.
- The component's listening port was not available as it is in use by something else.
- There was insufficient disk space on that host for the Sentinel Agent's or the component's logging.
- The component could not write to the specified log file due to user or file permissions issues.
- The component could not locate its essential runtime libraries. Either the component or the libraries it requires were manually moved, or the environment variables are set incorrectly on that host.
- You might have set important management and configuration parameters but forgotten to commit them.
- The component might appear not to have started even though in fact it has, due to a firewall blocking the port that the EMM console uses to communicate with the component. The firewall must be disabled or reconfigured to allow traffic on the port the component is configured to listen on.
- The Sentinel Agent on that host could not locate the component's binary. This means it was not configured correctly; normally this is due to it not having had valid component paths specified.

**Note:** All components started by the Sentinel Agent are started under the same user context (and therefore with the same privileges) as the user the Sentinel Agent itself was started as.

Therefore, if the Sentinel Agent was started as a service as the user `Local System`, all the components it starts will also start as the user `Local System`. Note that `Local System` is not a Windows Domain user, and therefore might not be allowed access to Domain resources. This might be particularly relevant to IAF adapters which link against custom user code and are likely to require access to various network services and resources.


If this situation is encountered, you need to change the user that the Sentinel Agent is started as, which can be done through **Control Panel, Administrative Tools, Services**, right click on **Apama Sentinel Agent**, select **Properties**, then the **Log On** tab. Choose the **This account:** radio button and enter the username and password of the user to use instead. You can specify a domain user by preceding the username with the domain as follows: `domain_name/username`. You will of course have to be an Administrator on the machine to be able to do any of this.

## Stop component

This option is only available if a component is selected in the Navigation Pane. It is equivalent to pressing the  button on the toolbar.


Choose this option to stop the currently selected component. The component's icon will change to indicate that the component is no longer running or available. Note that as the icon is only updated once EMM can no longer communicate with the component this can take a few seconds.

## Restart component

This option is only available if a component is selected in the Navigation Pane. It is equivalent to pressing the  button on the toolbar.


Choose this option to trigger a restart on the currently selected component. The component's icon will change to indicate that the component has stopped and then change again to indicate it has started up and is operational again.

## Clear all component logs

This option is equivalent to clicking the  button. If Apama Managed Components is selected in the Navigation Pane, this option clears the diagnostic logs for all the managed components. If a host is selected in the Navigation Pane, it clears the diagnostic logs for all the components on that host.

## Configuring components with the details pane

The contents of the Details Pane change according to what is currently selected in the Navigation Pane. When a component is added to a host, it is automatically selected in the Navigation Pane.

**Note:** As already described, you need to press **Commit Changes** (  ) before any changes you make to any panel on the Details Pane take effect. The values that are currently in effect are shown within the rightmost grayed out labels.

## Specifying paths and filenames in the Details Pane

On several of the component Details Pane panels you are asked to provide a path or filename, (for example to specify where logging information should be stored, or where an adapter configuration file should be loaded from).

It is recommended that *absolute paths* and filenames be used where possible – an absolute path being one that specifies the whole path, for example:

`c:\Document and Settings\My User\apama-work\logs\` (on Windows)

or

`/home/apama-work` (on UNIX)

In contrast, *relative paths* are incomplete, or just consist of a filename on its own, such as:

`New_Correlator_1.log`

### Important notes

- Unless otherwise stated, all paths are located on the file system of the remote host, on which the Sentinel Agent and managed components are running, rather than the local host where EMM is running.
- Filenames and paths should never be enclosed in quotes, even if they contain spaces.
- If a relative path is provided – as is the case by default for component log files – it is assumed to be relative to the component's `Alternative start-up directory` setting if one was configured, or to the *current working directory* of the Sentinel Agent on that host if not. See "[Working directory](#)" on [page 38](#) for details of how the Sentinel Agent's working directory is determined.

## Preferences

---

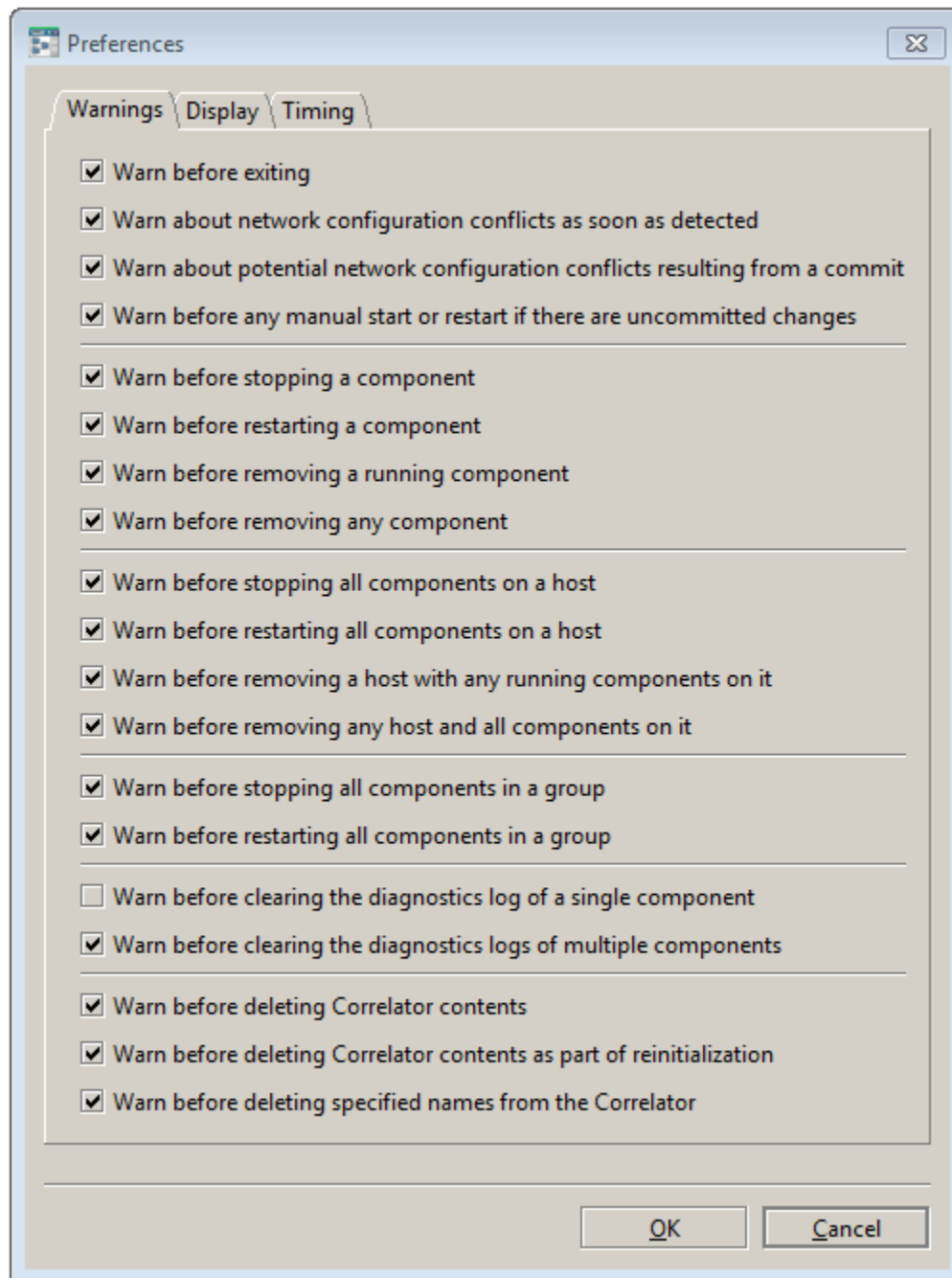
Several of EMM's default operational characteristics such as the warnings it issues, some of its display properties, and how it updates component information can be configured by the user.

These options are set using the Preferences dialog. This can be accessed by selecting the **Preferences...** option from the **Options** menu on the EMM menu.

The dialog has three tabs: **Warnings**, **Display**, and **Timing**.

## Warnings

The illustration below shows the **Warning** tab.



The **Warning** tab allows various confirmation dialogs to be turned off if desired.

Miscellaneous warnings:

- **Warn before exiting** – Ask for confirmation when the user asks to shut down EMM. Although all settings in EMM are preserved on shutdown and re-read the next time it is restarted, the background monitoring and automatic restart functionality only works while EMM is running.

- **Warn about network configuration conflicts as soon as detected** – Warn if a component conflict (same host-port combination) is detected.
- **Warn about potential network configuration conflicts resulting from a commit** – Ask for confirmation if a component conflict (same host-port combination) would result from a **Commit Changes** operation.
- **Warn before any manual start or restart if there are uncommitted changes** – Provide a warning when a component is started or restarted from EMM and changes have been made to the component's temporary working configuration that will be ignored when the component is started, because they have not yet been committed.

Component operation warnings:

- **Warn before stopping a component** – Ask for confirmation when the user tries to stop a component.
- **Warn before restarting a component** – Ask for confirmation when the user tries to restart a component.
- **Warn before removing a running component** – Ask for confirmation when the user tries to remove a component that is running, and provide the option of removing without stopping the component.
- **Warn before removing any component** – Always ask for confirmation when the user tries to remove a component, whether it is running or not.

Host operation warnings:

- **Warn before stopping all components on a host** – Ask for confirmation when the user tries to stop all components on a specific host.
- **Warn before restarting all components on a host** – Ask for confirmation when the user tries to restart all components on a specific host.
- **Warn before removing a host with any running components on it** – Ask for confirmation when the user tries asks to remove a host with any components on it that are known to be running; this provides the option of removing the host without stopping running components.
- **Warn before removing any host and all components on it** – Always ask for confirmation when the user tries asks to remove a host, whether it contains running components or not.

Group operation warnings:

- **Warn before stopping all components in a group** – Ask for confirmation when the user tries to stop all components in the `Apama Managed Components` group.
- **Warn before restarting all components in a group** – Ask for confirmation when the user tries to restart all components in the **Apama Managed Components** group.

Clearing log warnings:

- **Warn before clearing the diagnostics log of a single component** – Ask for confirmation before clearing the diagnostics log when a single component is selected.

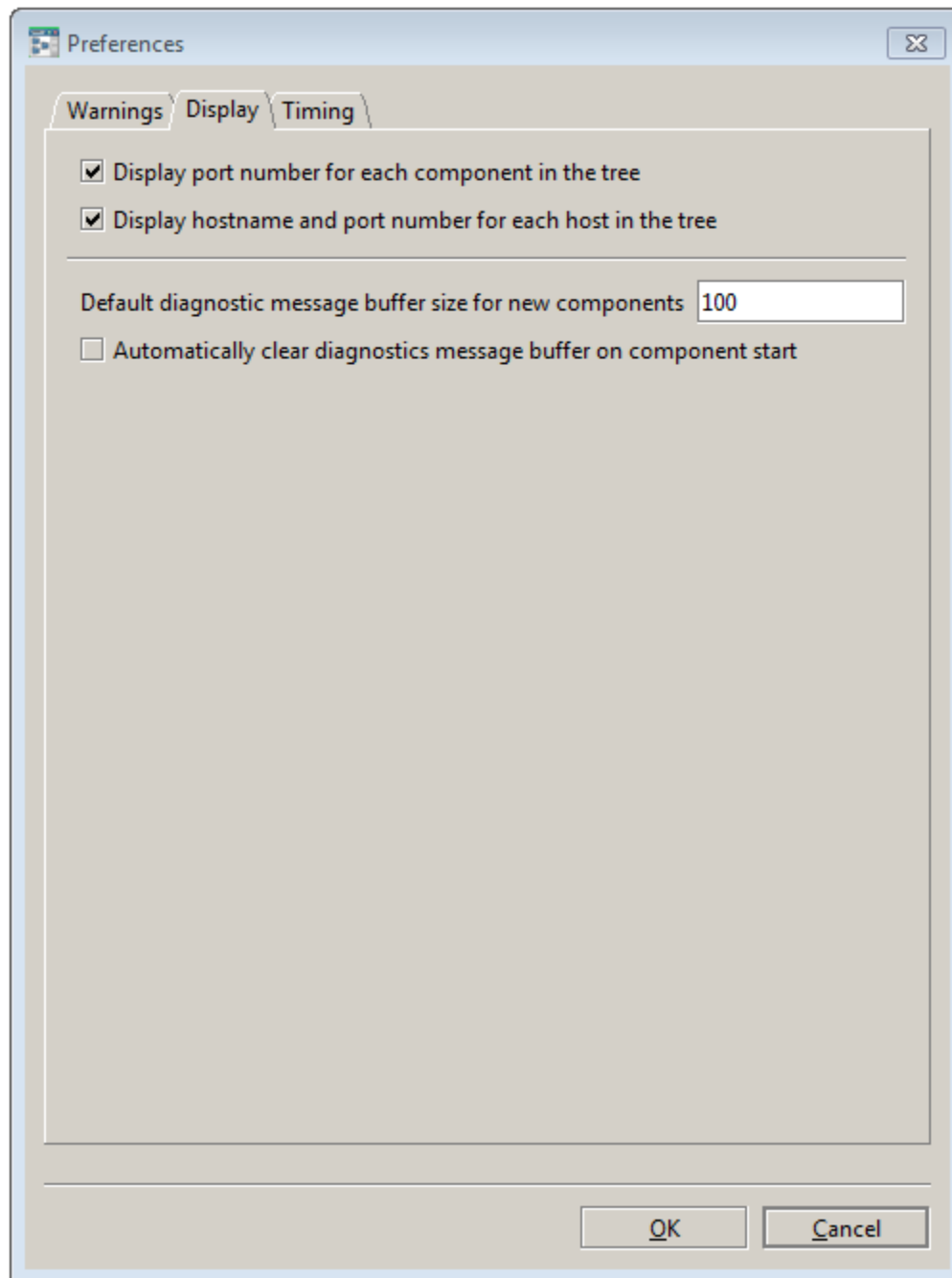
- **Warn before clearing the diagnostics log of multiple components** – Ask for confirmation before clearing the diagnostics logs for all the components associated with a selected host or group.

Deleting correlator contents warnings:

- **Warn before deleting Correlator contents** – Ask for confirmation before deleting all Apama Event Processing Language (EPL), Correlator Deployment Package (CDP), or JMon files in the correlator. The warning will be displayed after clicking the **Delete** button on the correlator's **Inspect** tab.
- **Warn before deleting Correlator contents as part of reinitialization** – Ask for confirmation before re-initializing the correlator with the **Delete everything loaded in the engine before reinitializing** check box is enabled.
- **Warn before deleting specified names from the Correlator** – Ask for confirmation before deleting specific EPL, CDP, or JMon files. The warning will be displayed after clicking the **Delete** button on the correlator's **Inspect** tab.

## Display

The illustration below shows the **Display** tab.



The following settings may be configured from the **Display** tab of the Preferences dialog:

- **Display port number for each component in the tree** – If enabled, the listening port for each component will be displayed alongside its name in the Navigation Pane. This makes it easier to check for conflicts (since the port number must be unique per host).
- **Display hostname and port number for each host in the tree** – If enabled, the actual hostname (as opposed to its EMM descriptive name) and the port its Sentinel Agent



is listening on are displayed alongside that host's name in the Navigation Pane. This makes it easier to avoid conflicts in host management.

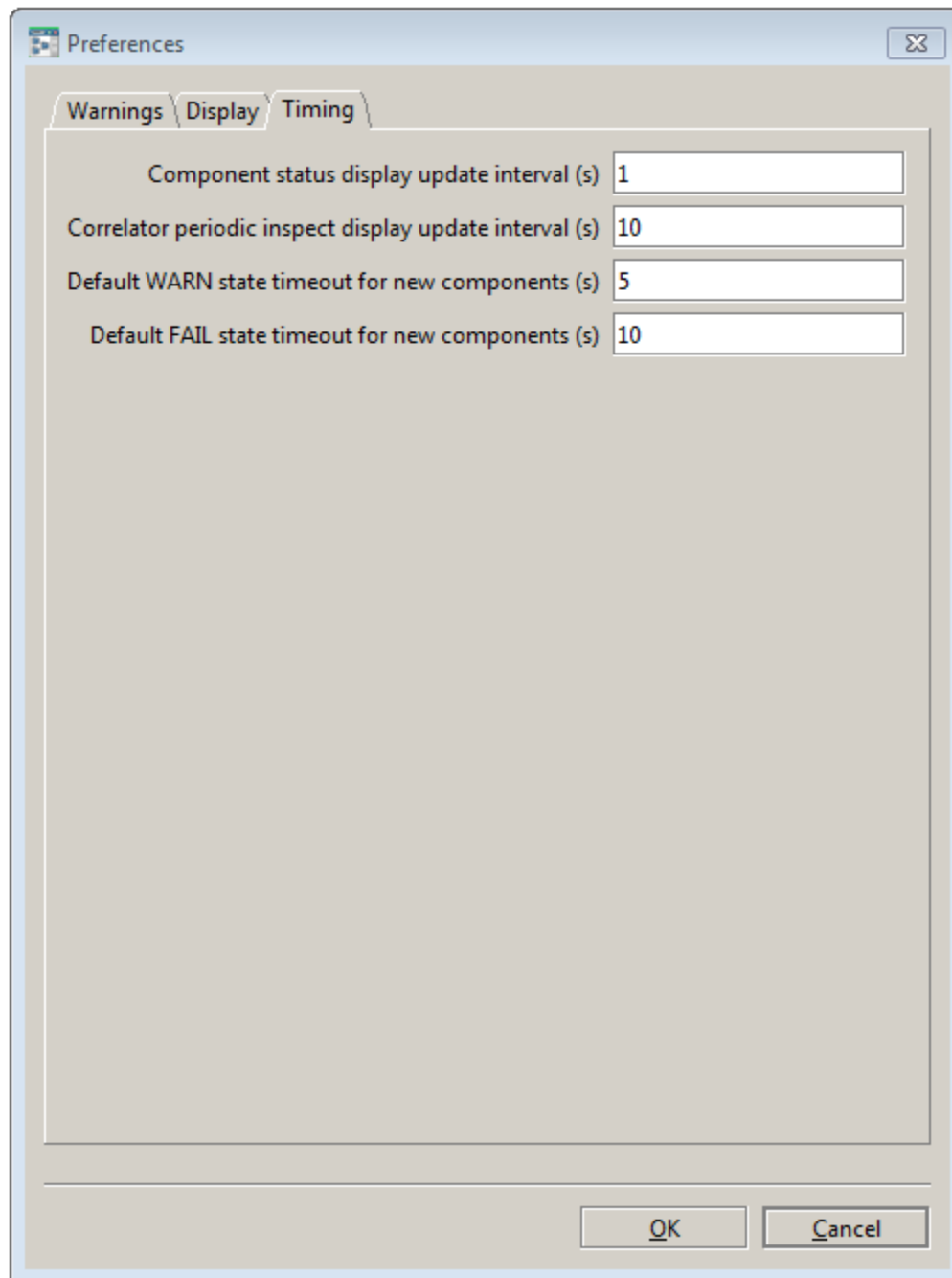
- **Default diagnostic message buffer size for new components** – Each component has a **Diagnostics** tab in its Display Pane, containing a list of the most recent status messages for the component. The maximum number of messages (after which the oldest will be discarded) may be configured on a per-component basis; this **Preferences** option specifies the default size of the message buffer that is assigned to new components. By default the buffer size is 100 lines. Minimum size is 1 line.
- **Automatically clear diagnostics message buffer on component start** – If enabled, removes all log messages for a component when it starts. This is useful during development work, but it should not be enabled in a production system.

## Timing

The illustration below shows the **Timing** tab.

The following settings may be modified from the **Timing** tab of the Preferences dialog:

- **Component status display update interval (s)** – This parameter specifies how frequently a component's statistics should be updated. By default this is every 1 second.
- **Correlator periodic inspect display update interval (s)** – This parameter specifies how frequently a correlator's inspect information should be refreshed. By default this is every 10 seconds.
- **Default WARN state timeout for new component (s)** – This parameter specifies the initial value of `WARN state timeout` used for new components (see ["Add correlator" on page 47](#)). By default this is 5 seconds.
- **Default FAIL state timeout for new component (s)** – This parameter specifies the initial value of `FAIL state timeout` used for new components (see ["Add correlator" on page 47](#)). By default this is 10 seconds.



## Deploying and Configuring Correlators


Apama correlators are the core event processing and correlation engines for Apama applications. This section describes how to start and manage correlators using the EMM console. You can also start and manage correlators using Apama command line utilities; for more information on this, see ["Correlator Utilities Reference" on page 129](#).

This topic describes how to use the EMM menu items to carry out operations on Apama correlators. However, all these operations can also be carried out from the context menus in the Navigation Pane, and the most useful are also available on the toolbar.

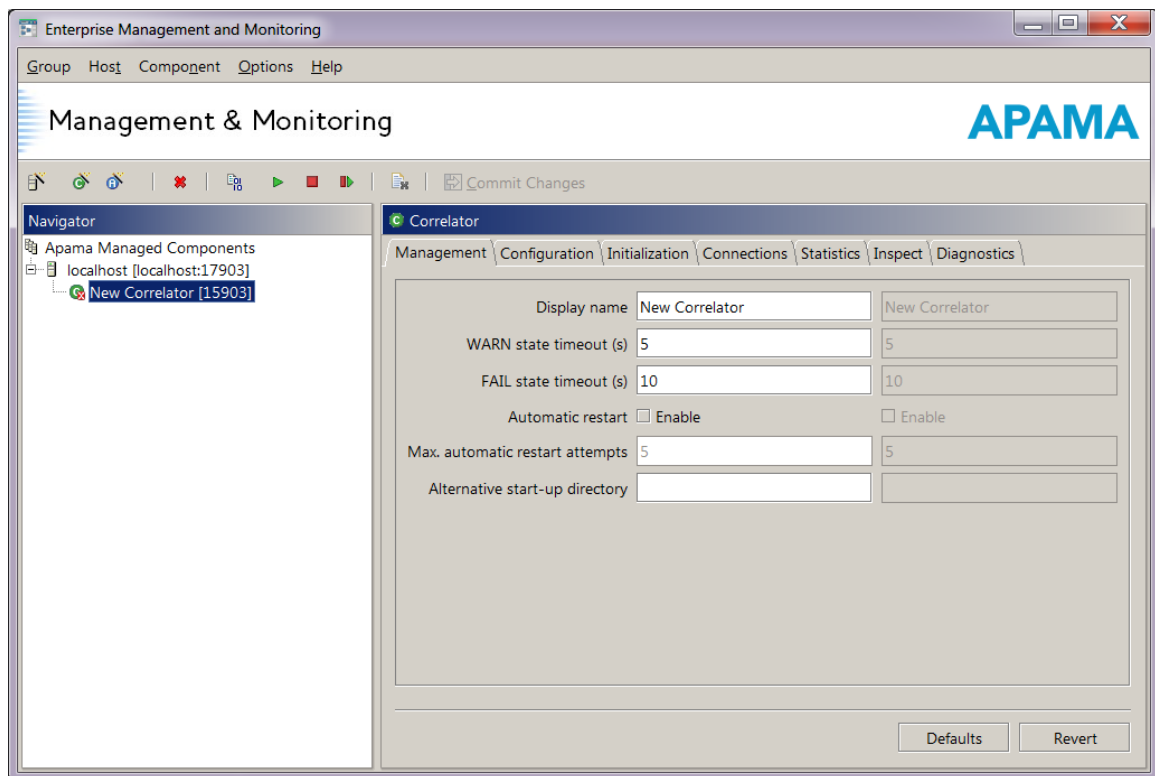
Before operations can be carried out on these components, a Sentinel Agent must be installed and running on the target host, and the EMM console must have been configured to manage that host, as detailed in ["Managing hosts" on page 35](#).

## Adding correlators

### To define and configure a new correlator


1. Select the host where you want to add the correlator in the EMM Navigation Pane.
2. Select **Component > Add correlator** from the EMM menu or click the  button on the tool bar

This adds a correlator to the host in the Navigation Pane, selects it, and displays the **Management** tab within the Details Pane;



The other tabs available on this Details Pane for a correlator are the **Configuration** tab, the **Initialization** tab, the **Connections** tab, the **Statistics** tab, the **Inspect** tab, and the **Diagnostics** tab. These are discussed in detail in ["The correlator tabs" on page 60](#).

EMM initializes the new correlator with a set of default options, which are normally safe. The default value for the listening port (which has to be unique per host) is automatically selected so as not to conflict with any other known components.

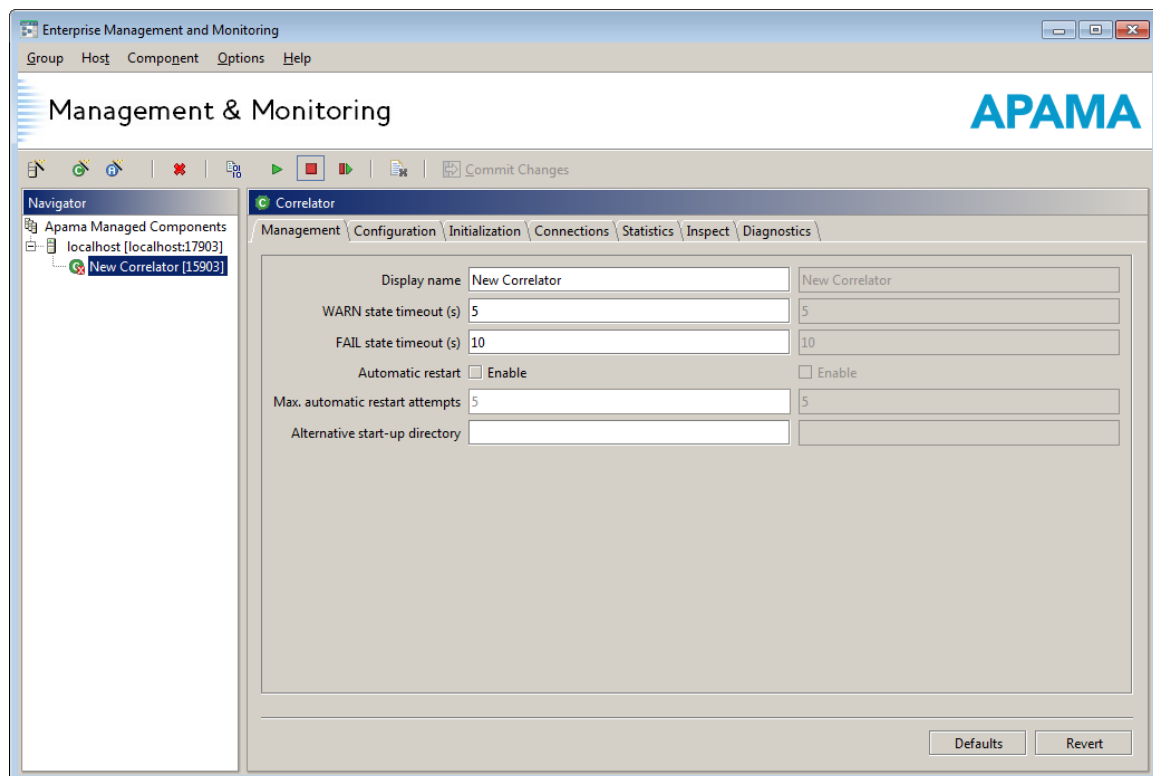
Before the new correlator can be started for this first time you must 'commit' its configuration using the **Commit Changes** (  ) button. Note that most of the correlator's configuration options only apply when the component is started up, so changes committed after the component is already running will usually not take effect until it is restarted.

## The correlator tabs

When a correlator is selected in the Navigation Pane, the Details Pane displays the tabs described in the topics below.

### Management tab

This tab contains a number of parameters that are relevant to managing a correlator.



The parameters on the **Management** tab are:

- **Display name** – This is the name to associate with this correlator in the Navigation Pane. The name can be changed at any time, even if the correlator has already been started.
- **WARN state timeout (s)** – This is the number of seconds to wait before moving the component into the `WARN` state if it is not changing state (stopping/starting/restarting) as required. This setting can be changed at any time. The default is 5 seconds.

- **FAIL state timeout (s)** – This is the number of seconds to wait after entering the `WARN` state before moving into the `FAIL` state, if the component has still failed to change state as expected. This setting can be changed at any time. The default is 10 seconds.
- **Automatic restart enable** – Tick to configure EMM to monitor the selected component. If it were to stop unexpectedly, EMM would automatically restart it – after waiting the amount of time required for it to enter the `FAIL` state (and subject to the configured limit on the number of restart attempts described below). This setting can be changed at any time.

**Note:** Component monitoring and auto-restart is carried out by EMM and requires EMM to be running.

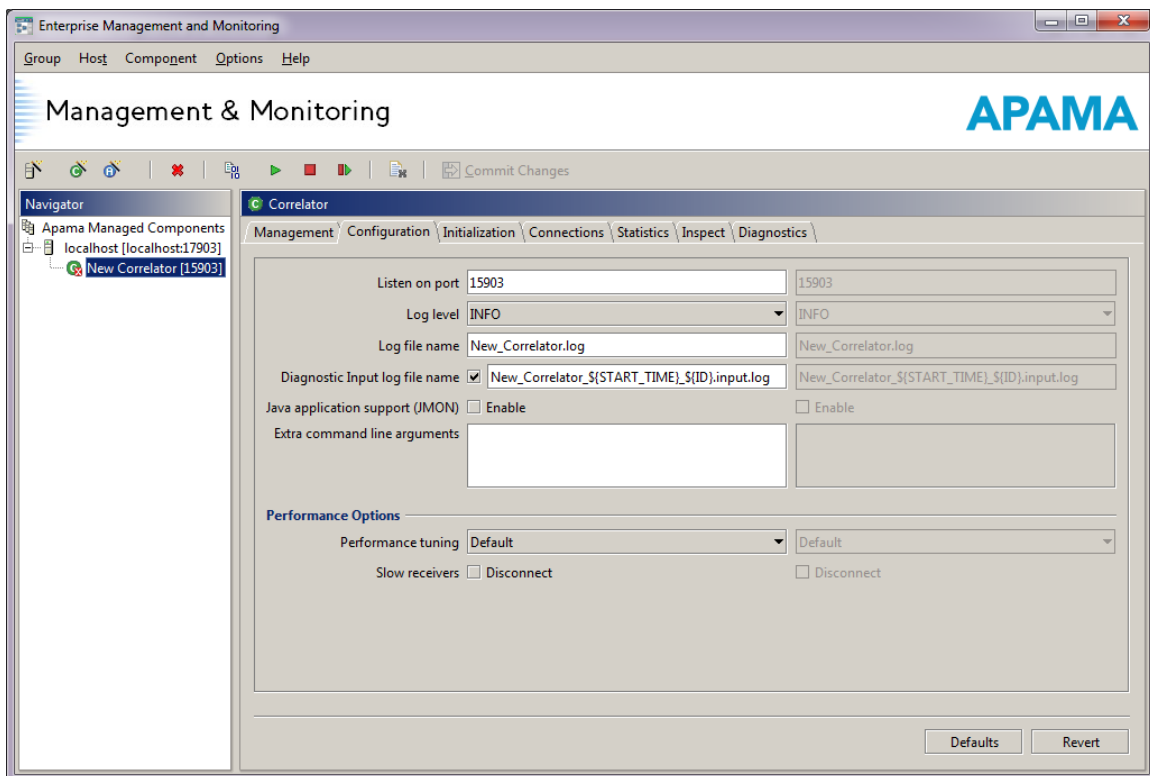
- **Max. automatic restart attempts** – This option is only available if **Automatic restart** is enabled. It specifies the total number of automatic restarts to perform (or attempt) without user intervention. The count is made from when you last enabled **Automatic restart** and committed, or explicitly stopped/started/ restarted the correlator. This setting can be changed at any time.
- **Alternative start-up directory** – By default all components are started from the current working directory of the Sentinel Agent running on the host in question (see ["Sentinel Agents" on page 35](#)). If you wish, you can provide an alternative startup folder here.

Click on the **Commit Changes** button when you are finished customizing the new correlator. This applies these outstanding changes.

The **Default** button resets all outstanding parameter values to their defaults, while the **Revert** button reverts the outstanding parameter values to their current committed values.

## Configuration tab

This tab contains a number of parameters that configure how a correlator operates. As all of these are startup parameters, you should adjust them before you start the component. If the component is already running they will only take effect the next time it is started.



The following parameters are available for a correlator:

- **Listen on port** – Port on which the correlator should listen for monitoring and management requests (default is 15903). The correlator will fail to start if this port is in use by any other component, or for that matter by any other software that is running on the relevant host.
- **Log level** – Sets the log level the event correlator should log at. This must be one of OFF, CRIT, FATAL, ERROR, WARN, INFO, DEBUG, or TRACE (in increasing order of verbosity). The default level is INFO.
- **Log file name** - Sets the filename that the event correlator should write log messages to (on the file system of the host the correlator is running on). It is recommended that an absolute path be provided. See ["Specifying paths and filenames in the Details Pane" on page 51](#) for more information about setting filename options correctly.
- **Diagnostic Input log file name** - Enables and sets the filename for a diagnostic input log. A diagnostic input log collects all messages sent to the correlator and writes then to a file on the file system of the host the correlator is running on. The input log can help Apama technical support diagnose problems with a correlator or an application running on the correlator.

It is recommended that an absolute path be provided. See ["Specifying paths and filenames in the Details Pane" on page 51](#) for more information about setting

filename options correctly. See also: ["Replaying an input log to diagnose problems" on page 216.](#)

- **Java application support (JMON)** – Enables support for JMon applications. If this is not set, any attempt to inject a JMon application either using `engine_inject -j` or by adding a `.jar` file `Inject` initialization action will result in an error. The correlator's performance is improved when Java application support is disabled.
- **Extra command line arguments** – This option allows additional unspecified command line arguments to be passed to the correlator. For example these might be special options to pass to the embedded JVM used for JMon applications, or special settings provided to you by Apama Customer Support to address specific issues.
- **Performance tuning** – Select **Slow receivers** to indicate that the correlator should disconnect any receiver whose output queue becomes full; that is, a receiver that cannot consume events fast enough. Without this option, the event correlator would eventually stop processing events altogether, (and block) until some output events are consumed by the receiver, freeing space on its output queue.

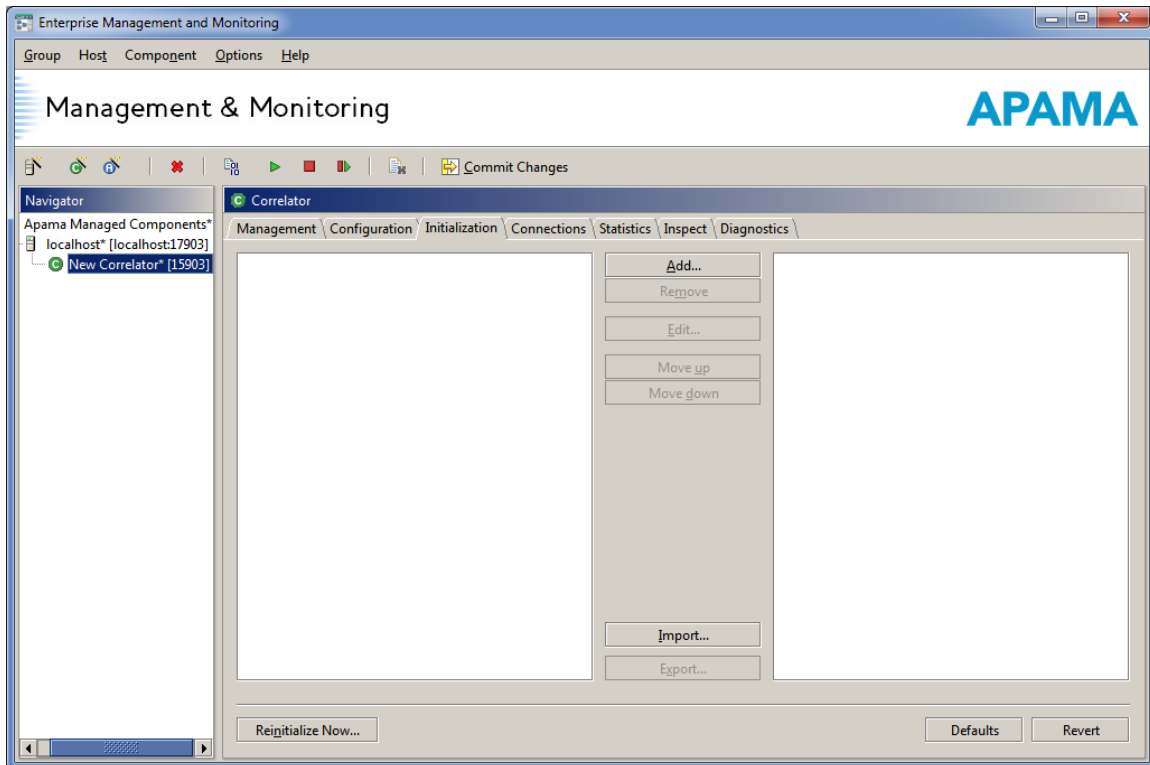
Click **Commit Changes** when you are finished customizing the new correlator. This applies these outstanding changes for use when the correlator is actually started.

The **Default** button resets all outstanding parameter values to their defaults, while the **Revert** button reverts the outstanding parameter values to their current committed values.

## Initialization tab

This tab allows you to specify the monitors and events that will be automatically executed whenever the component is started by EMM. The files you specify typically have `.mon`, `.evt`, and `.jar` extensions. Note that if the component is started outside EMM and then added to EMM for management, any initialization actions are only executed the next time it is started through EMM.

**Note:** By default, EMM uses the default encoding of the system on which EMM is running to read Apama Event Processing Language files (`.mon`) and event (`.evt`) files. If you want to inject UTF-8 encoded files, ensure that you use an editor that adds a Byte Order Mark (BOM) at the start of the file. For example, on Windows, the Notepad editor inserts a BOM when you save a file.

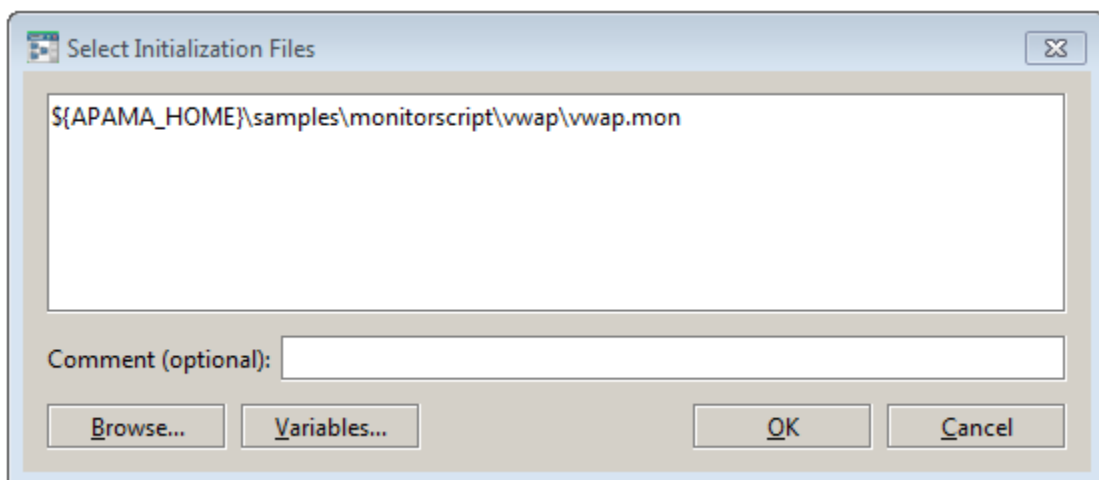


### Adding initialization actions

#### To add initialization actions

1. On the **Initialization** tab, click the **Add** button.

This displays the Select Initialization Files dialog.



2. Enter the name of the monitor or event file to use for initializing the component. You can also click **Browse** to navigate to the directory containing the files to use for initializing the component and select the files. Use CTRL and SHIFT to select more

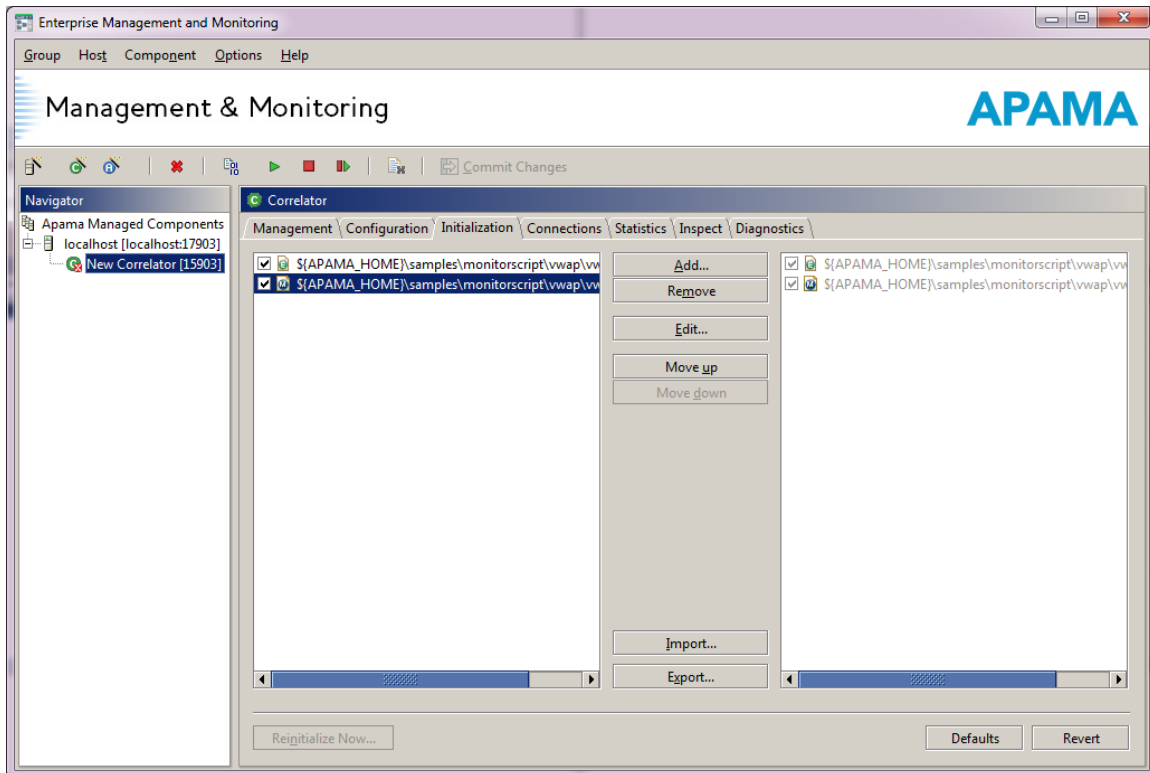


than one file name. You can add a comment that will be displayed next to the file name on the **Initialization** tab.

3. Click **Variables** to select an Apama variable; this alleviates the need to type a full path name. You can also use your own (non-Apama) environment variables by specifying them in the form: `${env_var:foo}`.
4. When you have specified the name of the file(s) you want to add, click **OK**.
5. You can also specify a file with a `.txt` extension containing a list of monitors and event files with which to initialize the component. Click **Import** and enter the name of the file that contains the initialization files. Files for this purpose are formatted with the name of each `.mon`, `.evt`, or `.jar` file on a separate line. Software AG Designer exports initialization files in this format.
6. Instead of entering or selecting file names as in Step 2, you can drag and drop files from a Windows Explorer window to the left hand list of the **Initialization** tab.

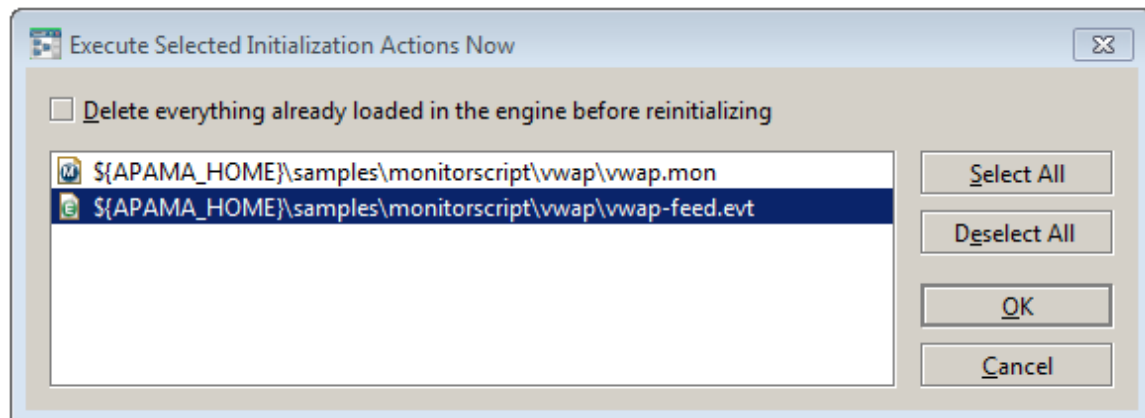
**Note:** All filenames specified here will be accessed on the local computer on which the EMM console is running – not the machine hosting the correlator (unless of course the correlator is also running on the local machine).

7. After the previous steps, EMM displays the initialization files in the left hand list. The check boxes in the left hand list allow you to select which initialization actions will take effect; actions with unchecked boxes will be ignored. Click the **Commit** button to apply the changes; this will display the initialization files in the right hand list. The initialization actions will take place the next time the component starts



When an action is selected, the following buttons are also available:

- **Remove** – Remove the selected initialization action.
- **Edit** – Allows the comment and/or filename associated with an initialization action to be changed.
- **Move up** – Move the selected initialization action up in the list. The order is significant because the actions will be executed in the order they appear in the list (from top to bottom).
- **Move down** – Move the selected initialization action down the list.
- **State restore** – Restore the entire runtime state of the correlator from a state image file or dump. Due to the nature of Restore operations, only one Restore initialization action may be specified for each correlator, and it will always be executed before any other Initialization action.
- **Import** – Allows you to specify a text file containing initialization actions.
- **Export** – Saves the component's initialization actions to a text file.
- **Reinitialize Now...** – Initializes the Component with the initialization files specified. When you click **Reinitialize Now...**, EMM displays the Execute Selected Initialization Actions Now dialog.

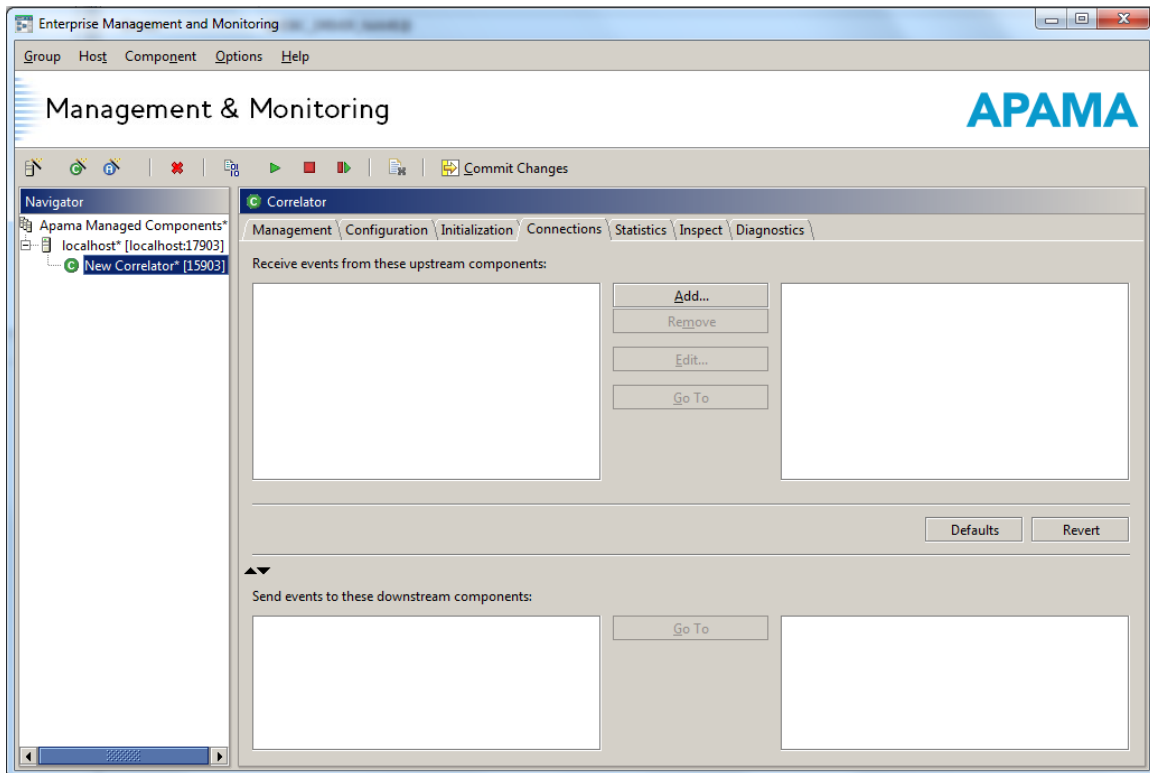


This dialog allows you to specify which actions you want to use and also gives you the opportunity to delete any Apama EPL or JMon files loaded in the correlator before you reinitialize it.

- **Defaults** — Removes any initialization files that have been added to the component.
- **Revert** — Removes initialization files that have been added to the component since the last time changes were Committed. Initialization files added prior to the last Commit are retained.

## Connections tab

This tab displays any other components that are connected to this component in both upstream and downstream directions. “Upstream” components are those components that this component receive events from; “downstream” components are those that this component sends events to. Because the “owner” of the connection is always the downstream component, the list of downstream connections is for display only.

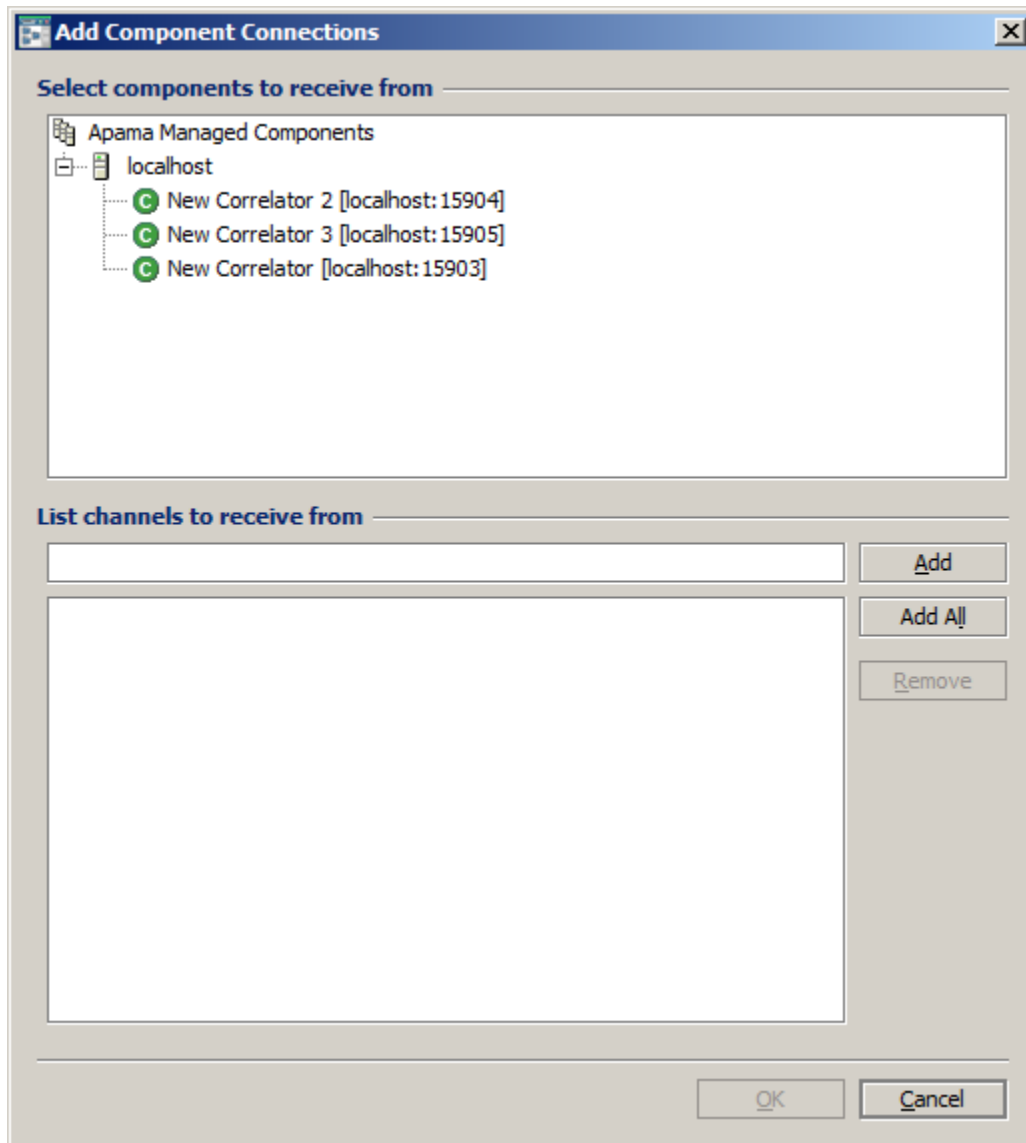


### ***Adding new upstream connections***

The **Connections** tab allows you to add new upstream component connections.


#### **To specify upstream components you want to connect to this component**

1. Click **Add** to display the Add Component Connections dialog.

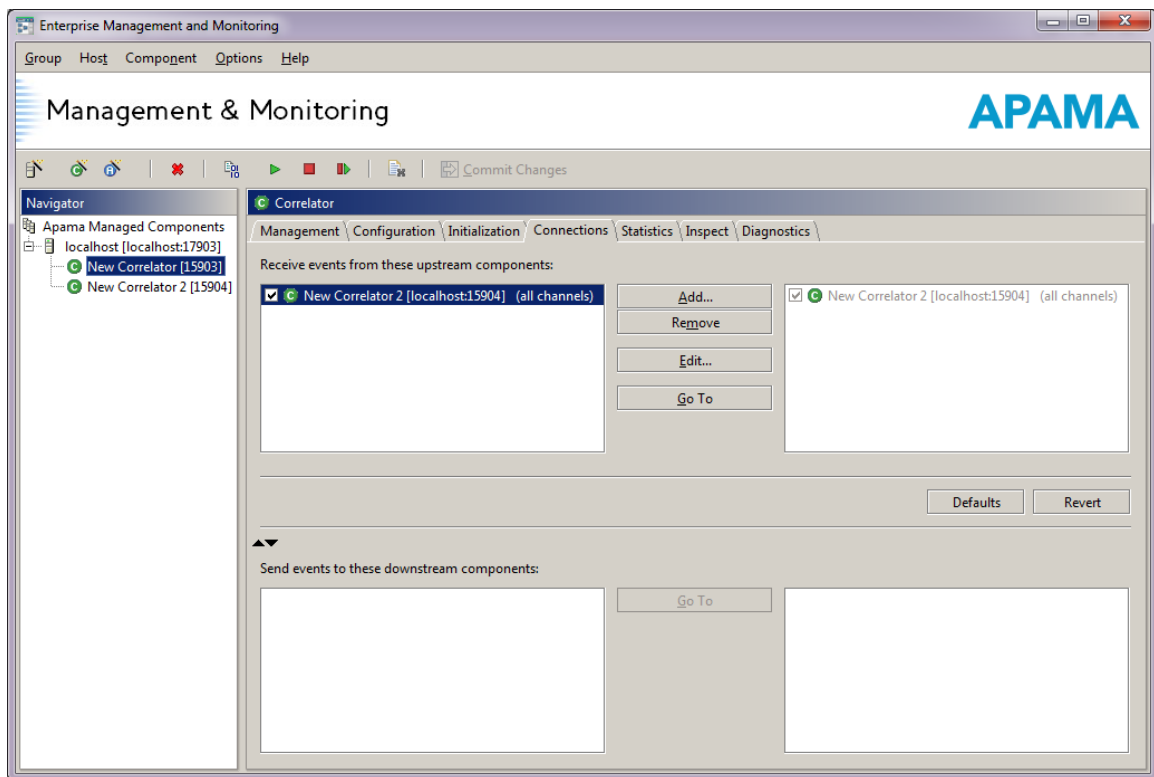


2. From the **Select components you want to receive from** list, select one or more components.
3. In the **List channels to receive from** field, enter the name of one or more channels associated with the component you selected in Step 2 and click **Add**, or click **Add All** to receive all channels associated with a component.
4. Click **OK**. The **Connections** tab will display the components you added in the left hand list.

The check boxes in the left hand list of the upstream connections allow you select which connections will take effect in the next step. Connections with checked boxes will be enabled; those with unchecked boxes will be disabled.

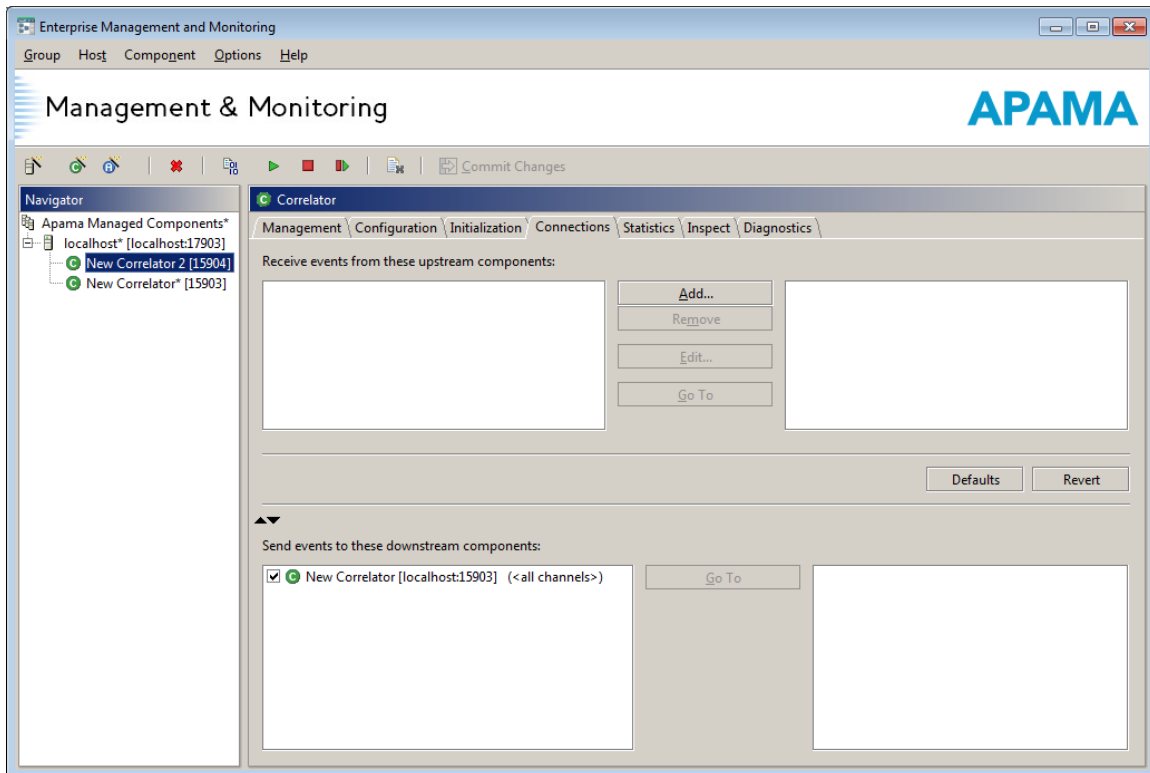
5. Select the **Commit changes** menu item or the click the **Commit Changes** (  ) button to make your changes take effect.

This adds the connected components to the right hand list of the Connections tab.



### ***Downstream connections***

Downstream components connected to this component are displayed in the **Send events to these downstream components** field. The following illustration shows a correlator with a downstream connection.



If you select an upstream or downstream component and click the corresponding **Go To** button, EMM switches the display to show the **Connections** tab for the connected component.

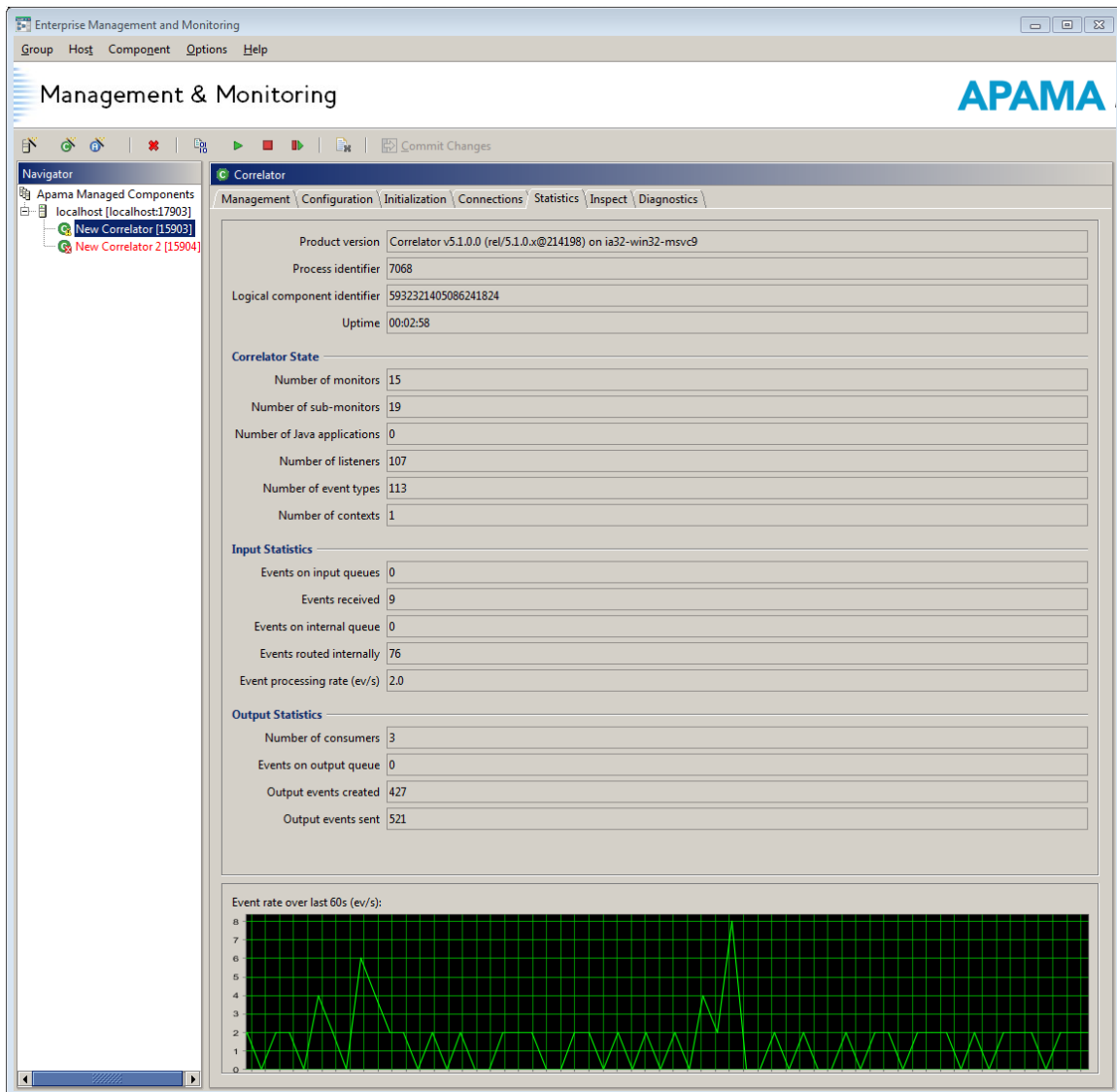
## Statistics tab

This tab allows you to monitor the status of an operational correlator.

Status information is refreshed regularly, by default, once a second. You can change this rate by changing the **Component status display update interval (s)** value on the **Timing** tab within the Preferences dialog. The Preferences dialog can be accessed from the **Preferences...** option from the **Options** menu.

No status information will be displayed if EMM cannot communicate with the selected correlator. If this is the case it is usually because the correlator has not yet been started or has been stopped. However, it could also be due to network failure. In this situation, statistics will reappear once the connection is restored.

You may need to scroll the display to view all the information on the **Statistics** tab. The following illustration shows the **Statistics** tab for an active correlator.



The **Statistics** tab displays the following information:

- **Product version** — The Apama release number.
- **Process identifier** — The identifier assigned to this component by the operating system.
- **Logical component identifier** — A numeric identifier for this component. This is useful for matching components with information contained in log files.
- **Uptime(s)** — The time in seconds since this correlator was started. This time is maintained and reported by the component itself, so if the correlator was started independently of EMM and only managed by EMM later, the value would still be accurate.



- **Number of monitors** – The current number of Apama monitors injected and instantiated inside the correlator. This figure changes upwards and downwards as monitors are injected, deleted or just expire.
- **Number of sub-monitors** – The number of Apama sub-monitors. Sub-monitors are created by 'spawn' actions within the monitor EPL code. This figure changes upwards and downwards as sub-monitors are spawned, killed or just expire.
- **Number of Java applications** – The number of JMon applications currently loaded into the correlator. JMon applications do not expire, so this value only decreases when they are explicitly unloaded.
- **Number of listeners** – The number of event listeners created by monitors and sub-monitors in EPL code and by JMon applications.
- **Number of event types** – The total number of event types defined within the correlator. This figure decreases when event types are deleted from the correlator.
- **Number of contexts** – The number of contexts in the correlator. This includes the main context plus any created contexts.
- **Events on input queue** – Across all contexts, the total number of events on input queues.
- **Events received** – The total number of events ever received by the correlator. This value is preserved by a checkpoint, so if the state of the correlator is restored from a checkpoint file it will reflect the total number of the events seen by the correlator from which the checkpoint was originally made.
- **Events on internal queue** – Across all contexts, the total number of routed events waiting to be processed. The internal routing queue is a high priority queue that is used when events are internally routed by the `route` instruction in EPL code or in JMon applications. For each context, the correlator processes events on the internal queue before processing other events on the input queue.
- **Events routed internally** – Across all contexts, the total number of events that have been routed since the correlator was started. This value is preserved by a checkpoint, so if the state of the correlator is restored from a checkpoint file it will reflect the total number of the events routed by the correlator from which the checkpoint was originally made.
- **Event processing rate (ev/s)** – The number of events per second currently being processed by the correlator. This value is computed with every status refresh and is only an approximation.
- **Number of consumers** – The number of event consumers registered with the correlator to receive events emitted by it.
- **Events on output queue** – The number of events waiting on the output queue to be dispatched to any registered event consumers.
- **Output events created** – The total number of output events created by the correlator. This value is preserved by a checkpoint, so if the state of the correlator is restored

from a checkpoint file it will reflect 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 dispatched to event consumers by the correlator. This figure varies from the preceding statistic as an output event might be sent to multiple event consumers. This value is preserved by a checkpoint, so if the state of the correlator is restored from a checkpoint file it will reflect the total number of output events sent out by the correlator from which the checkpoint was originally made.

A graph showing the values taken by the **Event rate** statistic over the last 60 seconds is displayed below the figures.

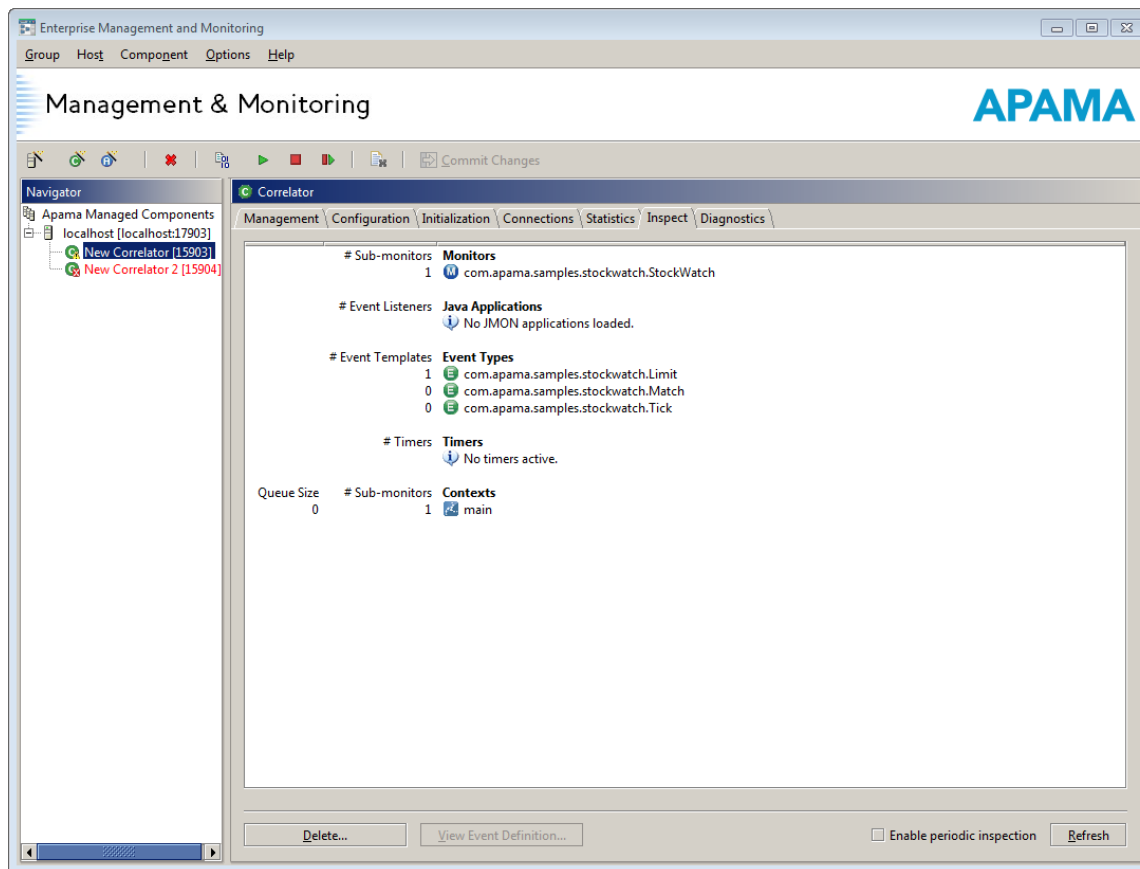
## Inspect tab

This tab displays the EPL monitors, JMon applications, and event types currently active within the correlator.

The information on the **Inspect** tab is retrieved in real time from the selected correlator when you actually select the **Inspect** tab, and it might take a few seconds to display.

You can configure the **Inspect** tab to refresh periodically by ticking the **Enable periodic inspection** checkbox at the bottom-left corner of the tab. By default the information will be refreshed every 10 seconds. You can change this value by changing the **Correlator periodic inspect display update interval(s)** value on the **Timing** tab within the Preferences dialog. The Preferences dialog can be accessed from the **Preferences...** option from the **Options** menu.

Alternatively you can refresh manually by clicking the **Refresh** button at the bottom-right corner. The following illustration depicts the **Inspect** tab for a correlator showing active definitions.



The information displayed is organized into the following groups: **Monitors**, **Java Applications**, **Event Types**, and **Contexts**.

- **Monitors** — Displays a listing of all the EPL monitors active in the correlator and the number of spawned sub-monitors (if any) for each one.
- **Java Applications** — Displays a listing of JMon applications loaded into the correlator. Against each one is also displayed the number of active listeners created by that application.
- **Event Types** — Displays a listing of all the event types the correlator knows about. Against each event type will also be displayed the number of event templates of that type in use at present in active listeners.
- **Timers** — Displays the current EPL timers active within the system. The four types of timers which may be displayed here are *wait*, *within*, *at*, and *stream*. The *stream* timers are those set up to support the operation of stream networks.
- **Contexts** — Display the names of the contexts in the correlator and for each correlator, the number of monitor instances (sub-monitors) and the queue size (number of events on the input queue) of each context.

The **Inspect** tab contains two other buttons:

- **Delete** — Delete an item or items from the correlator.

- **View Event Definition** — View the code for the definition of a selected Event Type.

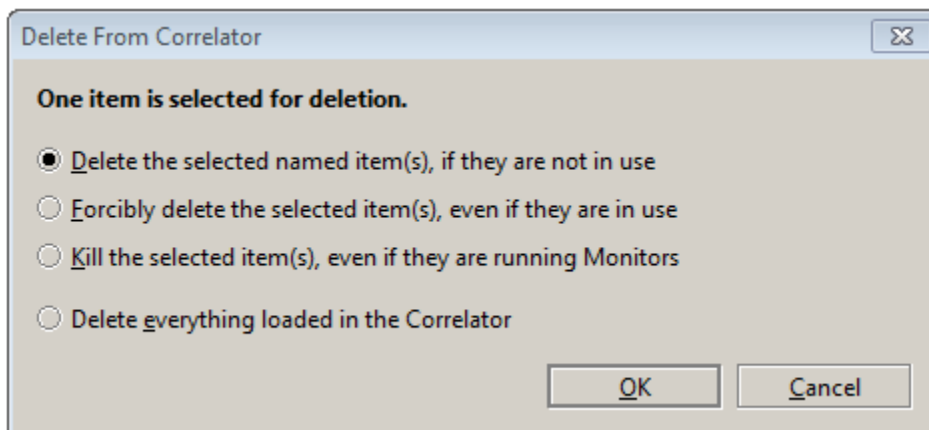
### *Deleting Monitors, Event Types, or JMon applications from a correlator*

To delete a monitor, event type, or JMon application from the correlator

1. In the **Inspect** tab, select the item or items you want to delete.
2. Click **Delete**.

This displays the Delete from Correlator confirmation dialog.

3. Select one of the following choices and click **OK**.
  - **Delete the selected named item(s), if they are not in use**
  - **Forcibly delete the selected item(s), even if they are in use**
  - **Kill the selected item(s), even if they are running Monitors**
  - **Delete everything loaded in the correlator**



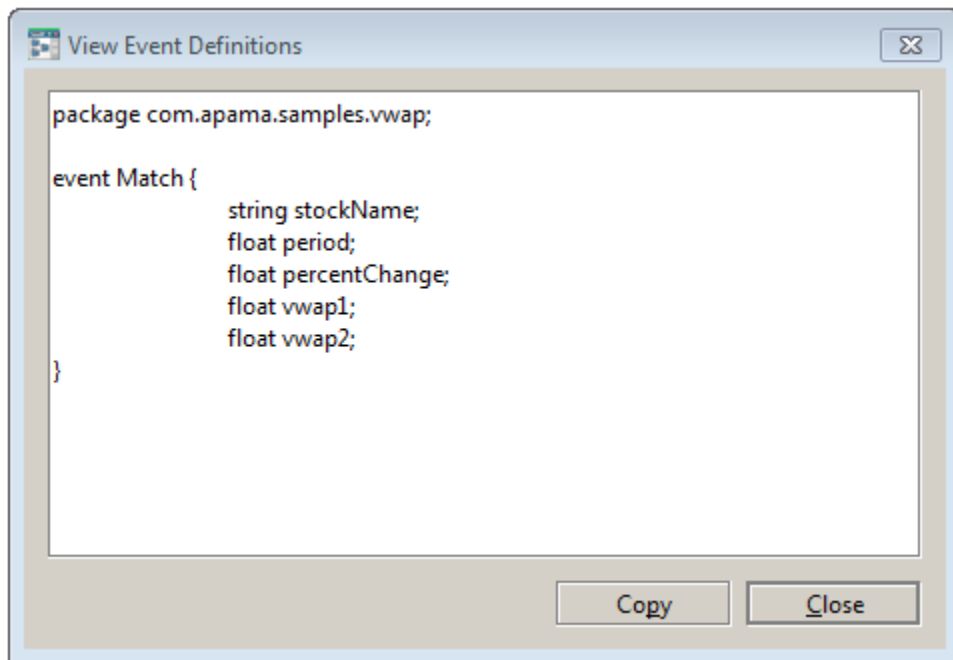
4. Select your choice and click **OK**.

### *Displaying event type definition for multiple Event Types*

To display the event type definition for one or more Event Types

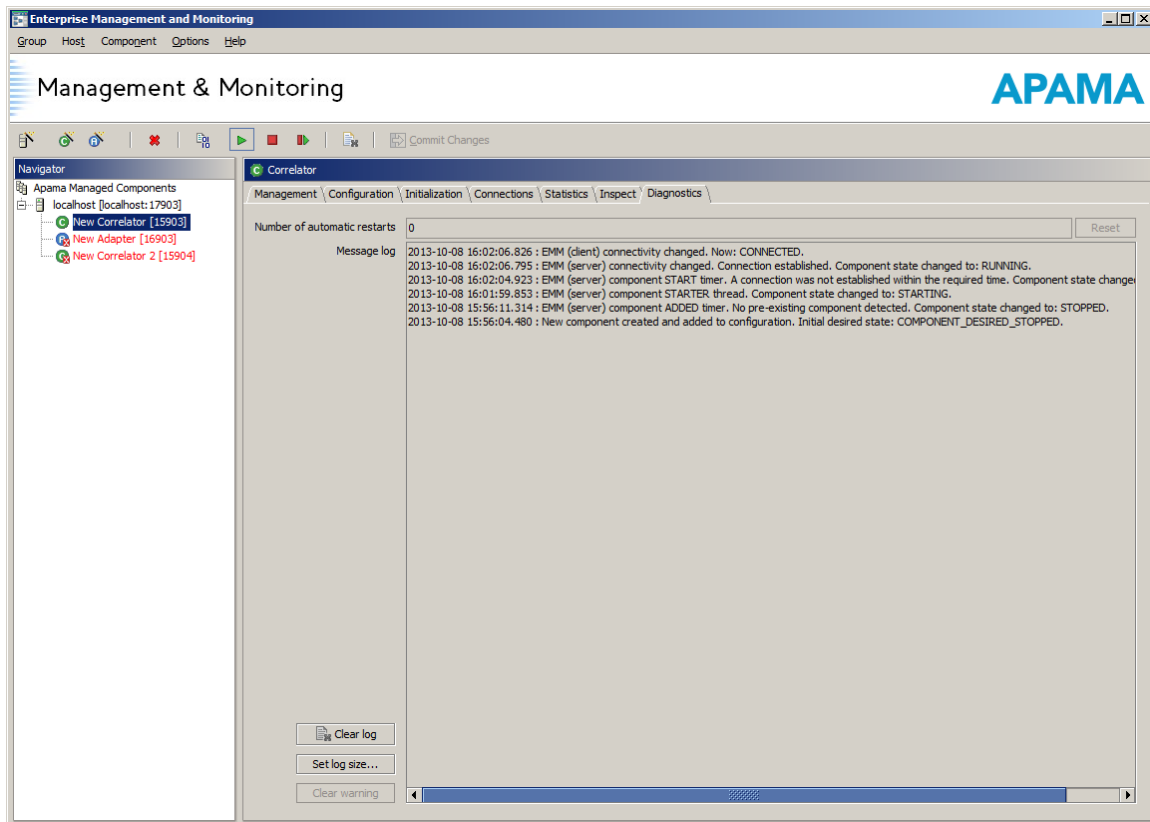
1. In the **Inspect** tab, select the Event Type or Event Types you are interested in.
2. Click **View Event Definition**.

This displays the View Event Definitions window.



## Diagnostics tab

The **Diagnostics** tab displays diagnostics information about the changes in the state of the correlator.



It provides two elements of diagnostics information:

- **Number of automatic restarts** – the total number of automatic restarts since the last time you manually started the component or since you enabled automatic restart. See ["Management tab" on page 60](#) for details of how automatic restart behavior can be configured. The **Reset** button lets you manually reset the counter and make EMM auto-restart the component again if it is currently in the FAIL state.
- **Message log** – A log of diagnostic information generated by EMM. Each log entry starts with the date and time of when the event being logged occurred. Note that most entries are tagged as being generated by EMM (server) or EMM (client). EMM (client) indicates the graphical front-end, whereas EMM (server) indicates the backend model.

The **Message log** has three associated buttons:

- **Clear log** – Removes all the messages logged for this component.
- **Set log size...** – Configures the size of the diagnostics log for this component. Note that the default log size assigned to new components can be configured from the Preferences dialog, and is set to 100 entries when Apama is installed. If the size is exceeded the oldest entries are removed to make way for new entries. Minimum size is 1 line.
- **Clear warning** – Changes the state of the component from WARN to RUNNING; see ["Component status indicators" on page 46](#) for a full explanation of the meaning

of the `WARN` state. This button is only enabled when the component is started and in the `WARN` state.

## Deploying and Configuring Adapters

Adapters allow Apama to interface with external sources of events like message buses, event feeds, or databases. You use the Integration Adapter Framework (IAF) with any of the standard, pre-packaged Apama adapters installed with the product or with custom adapters that you develop. To deploy an adapter, you specify the appropriate information in the adapter's configuration file and then start the IAF using the configuration file. This section describes how to deploy Apama adapters from the EMM console. You can also use Apama command line utilities to start and manage adapters. See also:


- "Using Standard Adapters" in *Connecting Apama Applications to External Components*
- "Developing Custom Adapters" in *Connecting Apama Applications to External Components*

This topic describes how to use the EMM menu items to carry out operations on Apama adapters. However, all these operations can also be carried out from the context menus in the Navigation Pane, and the most useful are also available on the toolbar.

Before operations can be carried out on these components, a Sentinel Agent must be installed and running on the target host, and the EMM console must have been configured to manage that host, as detailed in ["Managing hosts" on page 35](#).


## Adding adapters

### To define and configure a new IAF adapter

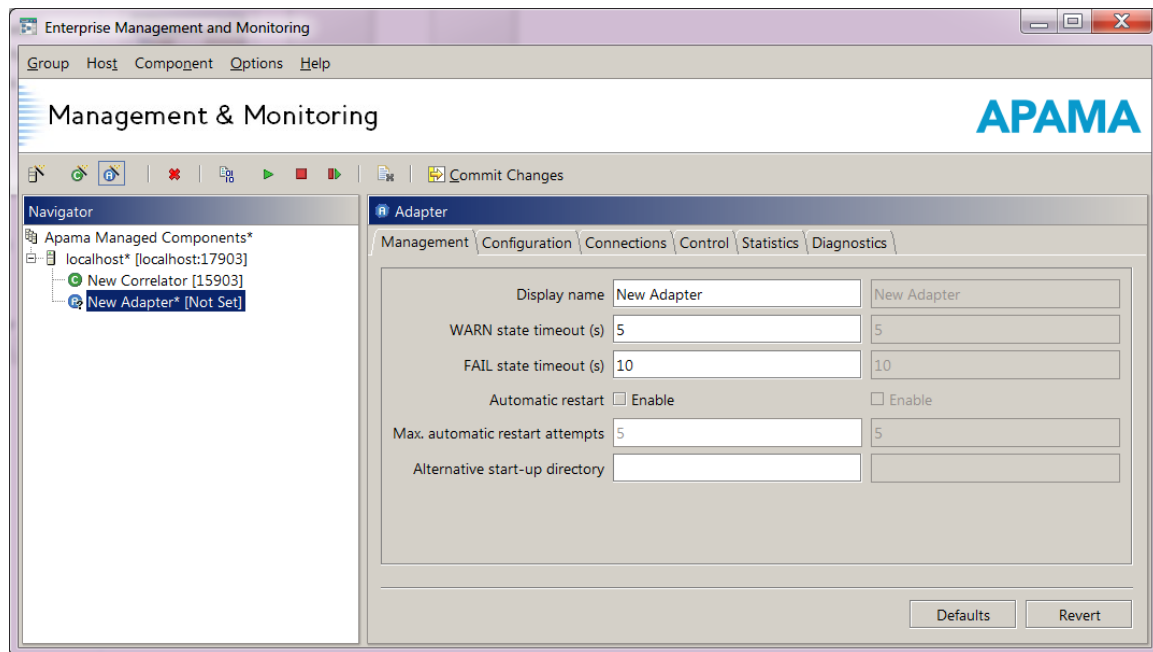
1. Select the host where you want to add the adapter in the EMM Navigation Pane.
2. Select **Component > Add adapter** from the EMM menu or click the  button on the toolbar.

This adds an adapter to the host in the Navigation Pane, selects it, and displays the **Management** tab in the Details Pane. The other tabs available on this Details Pane for an adapter are the **Configuration** tab, the **Connections** tab, the **Statistics** tab, the **Control** tab, and the **Diagnostics** tab. These are discussed in detail in ["The Adapter tabs" on page 81](#).

EMM initializes the new IAF adapter with a set of default options, which are normally safe. The default value for the listening port (which has to be unique per host) is automatically selected so as not to conflict with any other known components.


Before the new adapter can be started for this first time you must 'commit' its configuration using the **Commit Changes** () button. Note that most of the adapter's configuration options only apply when the component is started up, so changes

committed after the component is already running will usually not take effect until it is restarted.



## Configuring adapters

The contents of the Details Pane change according to what is currently selected in the Navigation Pane. When a component is added to a host, it is automatically selected in the Navigation Pane.

**Note:** As already described, you need to press **Commit Changes** (  ) before any changes you make to any panel on the Details Pane take effect. The values that are currently in effect are shown within the rightmost grayed out labels.

## Specifying paths and filenames in the Details Pane

On several of the component Details Pane panels you are asked to provide a path or filename (for example, to specify where logging information should be stored, or where an adapter configuration file should be loaded from).

It is recommended that *absolute paths* and filenames be used where possible – an absolute path being one that specifies the whole path, for example:

`C:\Users\Public\ApamaWork\logs\New_Correlator_1.log` (on Windows)

or

`/users/myusername/ApamaWork/logs/New_Correlator_1.log` (on UNIX)

In contrast, *relative paths* are incomplete, or just consist of a filename on its own, such as:

`New_Correlator_1.log`



**Important:** Unless otherwise stated, all paths are located on the file system of the *remote host*, on which the Sentinel Agent and managed components are running, rather than the local host where EMM is running.

Filenames and paths should never be enclosed in quotes, even if they contain spaces.

If a relative path is provided – as is the case by default for component log files – it is assumed to be relative to the component's **Alternative start-up directory** setting if one was configured, or to the *current working directory* of the Sentinel Agent on that host if not. See "[Working directory](#)" on [page 38](#) for details of how the Sentinel Agent's working directory is determined.

## The Adapter tabs

When an IAF adapter is selected in the Navigation Pane, the Details Pane will display the tabs described in the topics below.

### Management tab

This tab contains a number of parameters that are relevant to managing an IAF adapter, and is identical to that of a correlator.

The available parameters are:

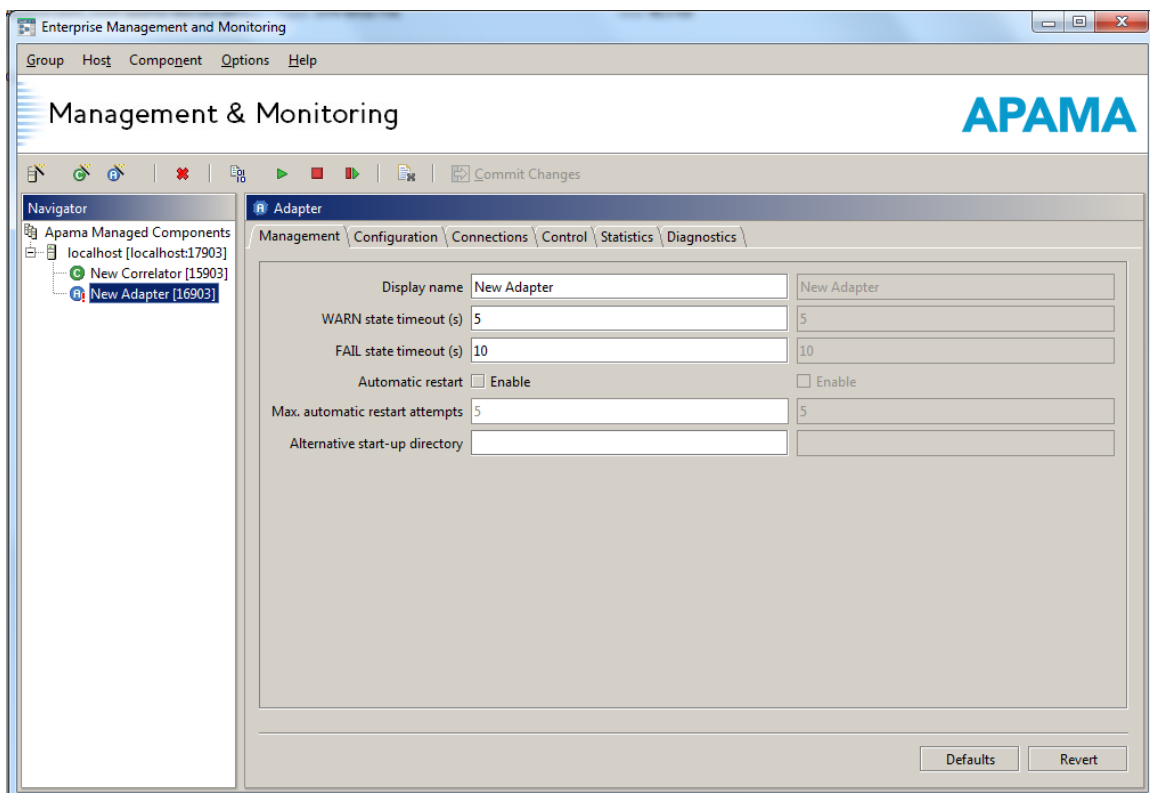
- **Display name** – This is the name to associate with this adapter in the Navigation Pane. The name can be changed at any time, even if the adapter has already been started.
- **WARN state timeout (s)** – This is the number of seconds to wait before moving the component into the `WARN` state if it is not changing state (stopping/starting/restarting) as required. This setting can be changed at any time.
- **FAIL state timeout (s)** – This is the number of seconds to wait after entering the `WARN` state before moving into the `FAIL` state, if the component has still failed to change state as expected. This setting can be changed at any time.
- **Automatic restart enable** – Tick to configure EMM to monitor the selected component. If it were to stop unexpectedly, EMM would automatically restart it – after waiting the amount of time required for it to enter the `FAIL` state (and subject to the configured limit on the number of restart attempts described below). This setting can be changed at any time.

**Note:** Component monitoring and auto-restart is carried out from EMM and requires EMM to be running.

- **Max. automatic restart attempts** – This option is only available if **Automatic restart** is enabled. It specifies the total number of automatic restarts to perform (or attempt) without user intervention. The count is made from when you last enabled **Automatic**

**restart** and committed, or explicitly stopped/started/ restarted the adapter. This setting can be changed at any time.

- **Alternative start-up directory** – By default all components are started from the current working directory of the Sentinel Agent running on the host in question (see in ["Working directory" on page 38](#)). If you wish, you can provide an alternative startup folder here. Note that this option is only functional if the Sentinel Agent running on the host in question is version 1.1.1 or greater.



Click the **Commit Changes** button when you are finished customizing the new IAF adapter. This applies the outstanding changes.

The **Default** button resets all outstanding parameter values to their defaults, while the **Revert** button reverts the outstanding parameter values to their current committed values.

## Configuration tab

This tab contains a number of parameters that configure how an IAF adapter operates. As these are all startup parameters, you should adjust them before you start the component. If the component is already running they will only have effect the next time it is started.

The following parameters are available for an IAF adapter:

- **Listen on port** - Port on which the IAF adapter should listen for monitoring and management requests (default is 16903). The adapter will fail to start if this port is in

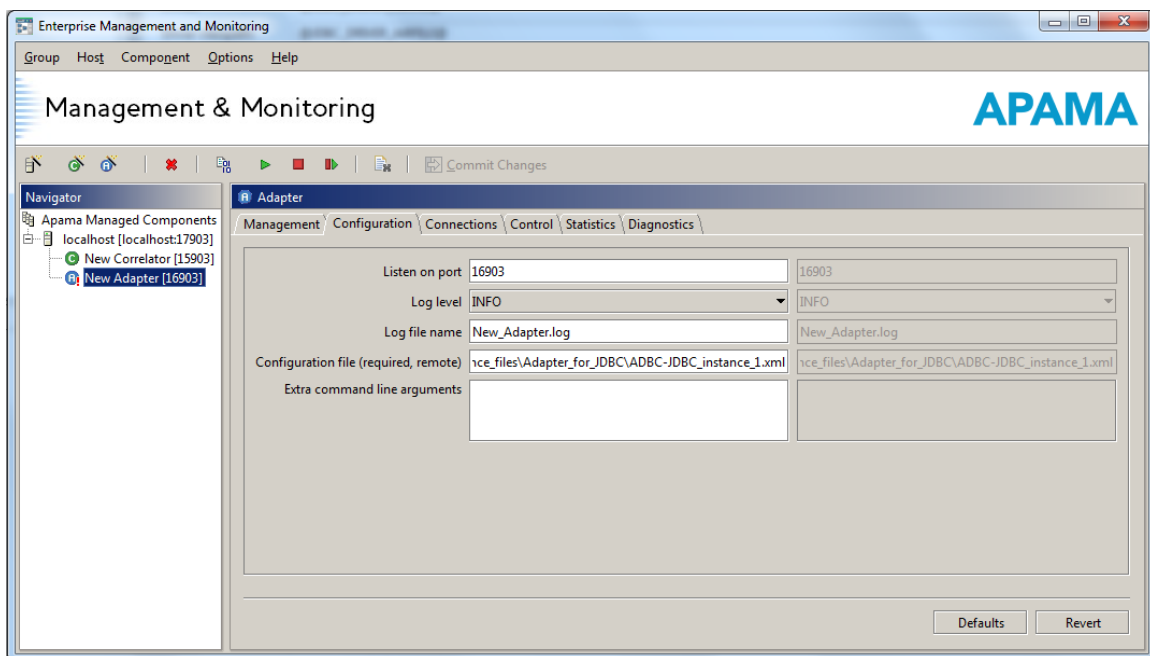
use by any other component, or for that matter by any other software that is running on the relevant host.

- **Log level** - Sets the log level the IAF adapter should log at – must be one of `CRIT`, `ERROR`, `WARN`, `INFO`, `DEBUG` (in increasing order of verbosity). If this value is not set, then the IAF adapter will log at the level specified in the adapter configuration file. If a value is indeed specified, that the EMM value will always override the log level specified in the adapter configuration file.
- **Log file name** - Sets the filename that the adapter should write log messages to (on the file system of the host the correlator is running on). It is recommended that an absolute path be provided.
- **Configuration file (required, remote)** – Specifies the path of an IAF adapter configuration file. The IAF adapter cannot be committed or started until a configuration file is specified.

For more information, see *The IAF configuration file* in *Connecting Apama Applications to External Components*.

The file's path is looked up on the file system of the host the adapter is running on, and it is recommended that an absolute path be provided. See ["Specifying paths and filenames in the Details Pane" on page 80](#) for more information about setting filename options correctly.

- **Extra command line arguments** – This option allows additional unspecified command line arguments to be passed to the IAF adapter. For example these might be special settings provided to you by Apama Customer Support to address specific issues.



Click on the **Commit Changes** button when you are finished customizing the new IAF adapter. This applies these outstanding changes for use when the adapter is actually started.

The **Default** button resets all outstanding parameter values to their defaults, while the **Revert** button reverts the outstanding parameter values to their current committed values.

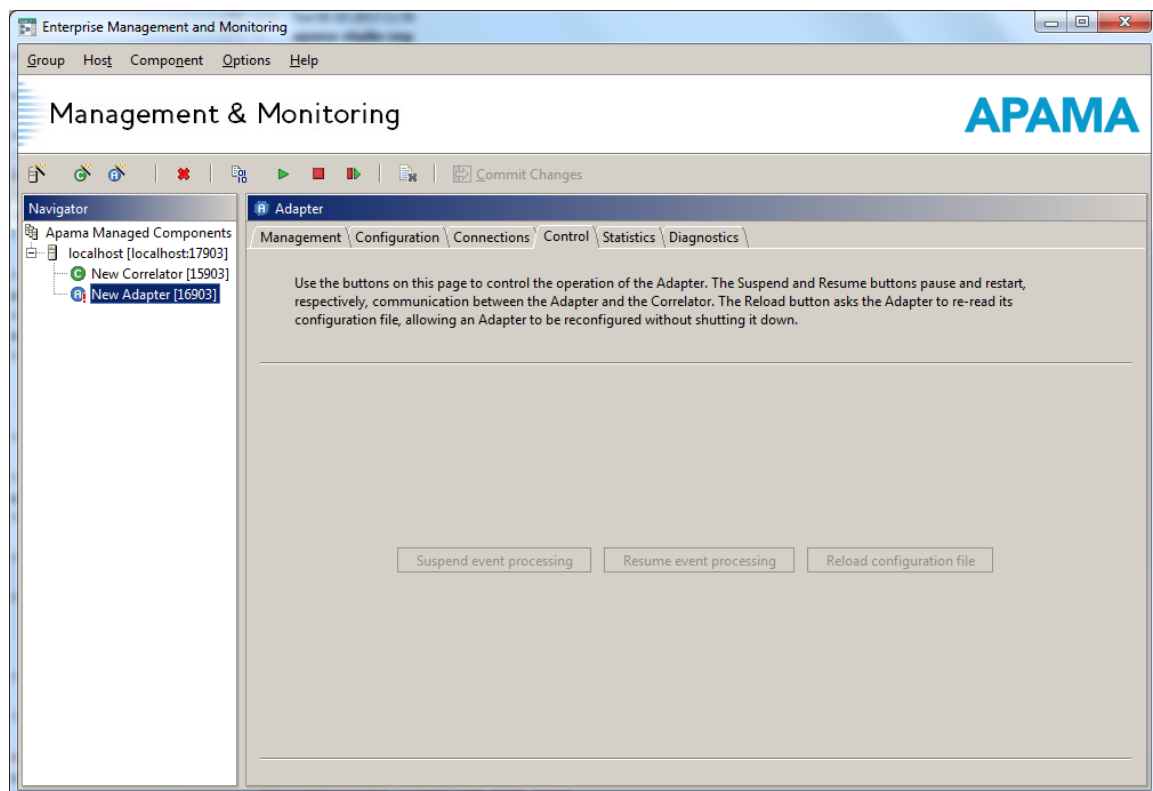
## Connections tab

This tab allows you to connect an adapter to upstream components. It also displays downstream components that are connected to the adapter. The tab is similar to the Connections tab for a correlator. For a complete description of the **Connections** tab, ["Connections tab" on page 67](#).

## Control tab

The **Control** tab provides for control of an IAF adapter. Three operations are supported:

- **Suspend event processing** – This button will pause a running IAF adapter.
- **Resume event processing** – This button will resume a running IAF adapter.
- **Reload configuration file** – This button will ask the IAF adapter to re-read its configuration file, allowing an adapter to be reconfigured without shutting it down.



## Statistics tab

This tab allows monitoring of the status of an operational IAF adapter.

Status information is refreshed regularly, by default, once a second. You can change this rate by changing the **Component status display update interval (s)** value on the **Timing** tab within the Preferences dialog. The Preferences dialog is available by selecting **Options > Preferences** from the EMM menu.

No status information will be displayed if EMM cannot communicate with the IAF adapter selected. This is usually because the adapter has not yet been started or has been stopped. However, it could also be due to network failure. In this case, statistics will reappear once the connection is restored.

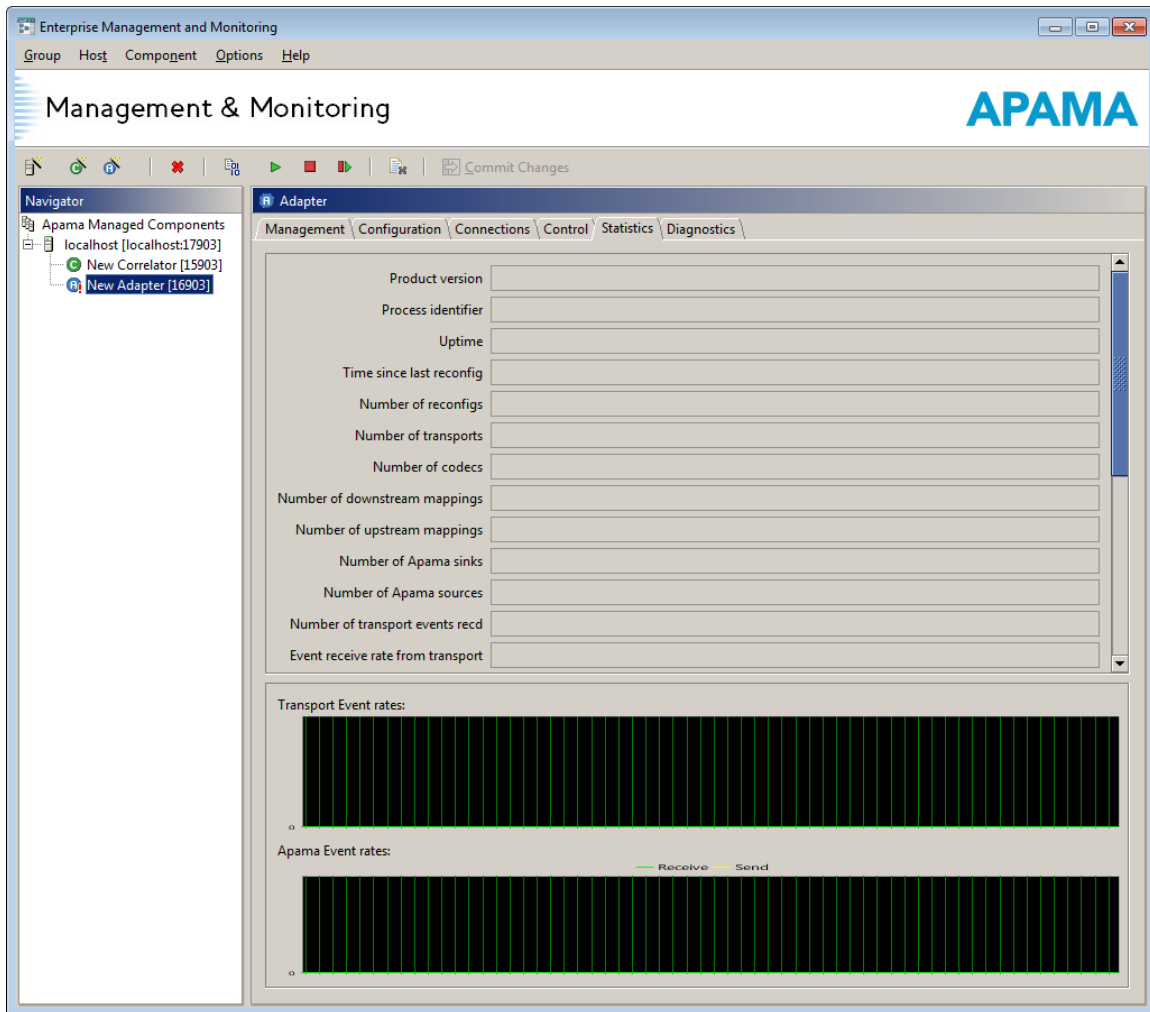
The following statistics are available:

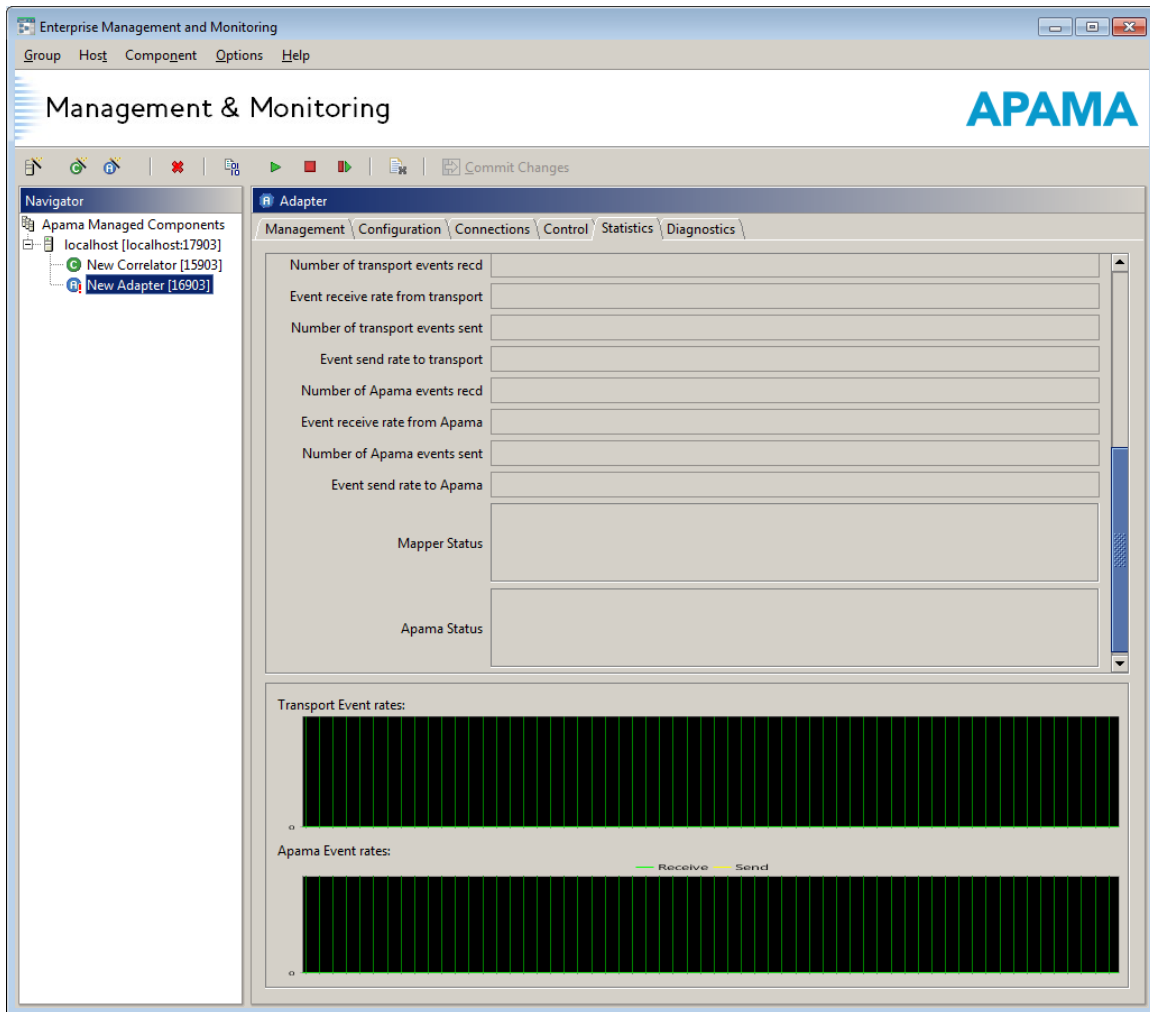
- **Product version** – The Apama release number.
- **Process identifier** – The identifier assigned to this component by the operating system.
- **Uptime** – The time in seconds since this adapter was started. This time is maintained and reported by the component itself, so if the adapter was started independently of EMM and only managed by EMM later, the value would still be accurate.
- **Time since last reconfig** – The time in milliseconds since this adapter was last 'reconfigured', that is, reset with a new configuration file.
- **Number of reconfigs** – The total number of reconfigurations.
- **Number of transports** – The total number of Transport Plug-ins active in this adapter.
- **Number of codecs** – The total number of Codec Plug-ins active in this adapter.
- **Number of downstream mappings** – The total number of mapping rules set up in the adapter's semantic mapper for downstream (from an external message source into Apama events) event mappings.
- **Number of upstream mappings** – The total number of mapping rules set up in the adapter's semantic mapper for upstream (from Apama events into an external message sink) event mappings.
- **Number of Apama sinks** – The number of Apama correlators that the adapter is sending events to.
- **Number of Apama sources** – The number of Apama correlators that the adapter is receiving events from.
- **Number of transport events recd** – The total number of messages received by the adapter's transport Plug-ins from external message sources (that is, downstream).
- **Event receive rate from transport** – The number of messages per second currently being received by the adapter's transport plug-ins. This value is computed with every status refresh and is only an approximation.
- **Number of transport events sent** – The total number of messages sent out by the adapter's Transport Plug-ins to external message sinks (i.e. upstream).

- **Event send rate to transport** – The number of messages per second currently being sent out by the adapter's Transport Plug-ins. This value is computed with every status refresh and is only an approximation.
- **Number of Apama events recd** – The total number of Apama events received by the adapter from Apama for upstream conversion.
- **Event receive rate from Apama** – The number of Apama events per second currently being received by the adapter from Apama. This value is computed with every status refresh and is only an approximation.
- **Number of Apama events sent** – The total number of Apama events sent to Apama by the adapter because of a downstream conversion.
- **Event send rate to Apama** – The number of Apama events per second currently being sent to Apama by the adapter. This value is computed with every status refresh and is only an approximation.
- **Mapper Status** – Indicates whether the adapter's semantic mapper has started.
- **Apama Status** – Indicates whether the adapter has contacted the Apama sinks and sources specified in the Configuration File.

The four event rate statistics are also displayed graphically over the last 60 seconds at the bottom of the **Statistics** tab in the two graphs: **Transport Event rates** and **Apama Event rates**.

The following two illustrations show the top and bottom portions of the **Statistics** tab for adapters.

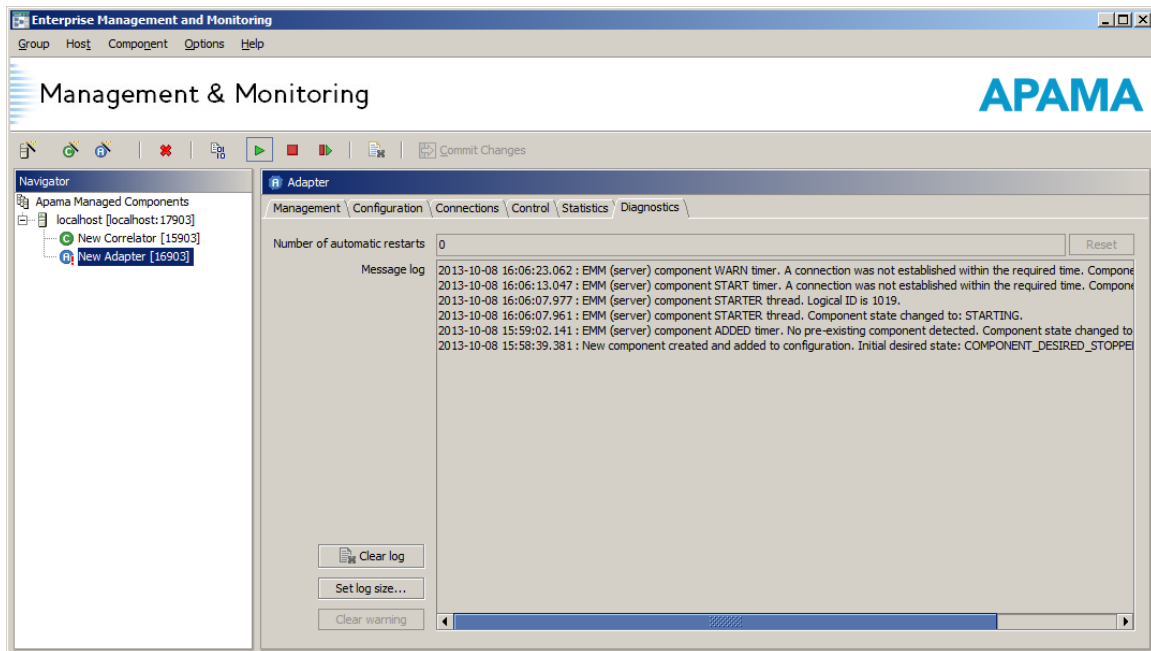




## Diagnostics tab

The Diagnostics tab for an IAF adapter is identical to that of a correlator and displays diagnostics information about changes in the state of the adapter.





For complete information on this tab, see ["Diagnostics tab"](#) on page 77.



# 4

## Deploying and Managing Queries

---

- Overview of deploying and managing query applications ..... 92
- Query application architecture ..... 92
- Deploying query applications ..... 93
- Running queries on correlator clusters ..... 94
- Managing parameterized query instances ..... 98
- Monitoring running queries ..... 99

As mentioned elsewhere, scaling, both vertically (same machine) and horizontally (across multiple machines), is inherent in Apama query applications. Scaled deployments on multiple machines use distributed cache technology to maintain and share application state. Consequently, deployment of Apama query applications includes setting up a distributed cache as well as some kind of messaging. The topics in this section provide instructions for doing this with the recommended platforms.

## Overview of deploying and managing query applications

---

Typically, query application deployments script the start up and management of all Apama query application components outside of the Apama's development environment in Software AG Designer. Apama recommends the use of the Ant export facility of Software AG Designer to aid in this.

Queries can also be run from Software AG Designer. However, Software AG Designer can run only a single correlator deployment. Use Apama macros for Ant to deploy multiple correlator deployments.

Queries can be deployed on a single node, but typically would be deployed across multiple nodes, forming a cluster. While involving more components, a cluster provides:

- Scale out across multiple hosts
- Resiliency against failures
- Continued availability if some nodes fail

Using a cluster will involve the following:

- Some number of correlators that are executing queries
- A BigMemory distributed cache for storing event history
- A JMS bus for distributing events to correlators

There are likely to also be adapters or a message bus to deliver events generated by queries, and possibly adapters connected to each correlator. See ["Deploying and Configuring Adapters" on page 79](#).

Queries use the store name "ApamaQueriesStore". If you want to use BigMemory, you need to configure the BigMemory driver to use that as the store name, else an in-process only memory store will be used. See ["Configuring BigMemory driver" on page 95](#).

## Query application architecture

---

In a query deployment, incoming events are delivered to correlators, typically via a JMS message bus, such that every event is delivered to one correlator. The correlators store the event history for each query in BigMemory, a distributed cache. On every event, one correlator reads the latest history for the partition or partitions to which the event

belongs, and writes that event to the distributed cache for access by other correlators. The entire window history is then evaluated against the query patterns.

Queries can make use of the following technologies to provide a scalable platform:

- JMS queues — these are used to distribute events to multiple correlators, which automatically spreads the load across a number of servers.
- BigMemory distributed cache — this allows state (event history) to be accessed quickly across multiple servers, and replicated to safeguard against hardware failures. This should be configured to give the desired amount of resiliency and scaled appropriately to the deployment.

It is possible to use Apama queries in a standalone mode on a single correlator. This allows easy testing by means of event files. However, all state is stored in-memory, and is lost when the correlator is stopped. Thus, this mode is only recommended for development, not for deployments.

When an event is sent to a cluster of correlators over a JMS queue, this is what happens:

1. Each event goes to one correlator.
2. A received event is handled by one of several processing threads within that correlator.
3. The key of the event is extracted based on the definitions of running queries that use that event.
4. The window of events for that key value is retrieved from the distributed cache.
5. The current event is added to the retrieved window, which is written back to the cache.
6. The event pattern of interest (what you are looking for) is evaluated against the stored window to determine whether there is a match.

Because events are sent to multiple threads in different correlators, small differences in timing across hosts can result in events being processed out of order. If there are a large number of events in the window, the cost of reading and writing the historic window will be excessive. Events for the same key may be processed by different correlators. Consequently, between events, the only state kept by the system is the window of historic event data.

Upon matching an event pattern, queries may send events to other monitors or to adapters. These can be shared adapters across the cluster, or more typically, adapters local to each correlator.

## Deploying query applications

---

Apama recommends that you use the Ant export facility in Software AG Designer to help you deploy your query application. The general steps for deploying an Apama query application include:

1. In Software AG Designer, configure JMS bus access and distributed cache access. See "Correlator arguments" in *Using Apama with Software AG Designer*.
2. In Software AG Designer, generate an Ant deployment script. The generated files are placed in a directory that you specify. See "Exporting to a deployment script" in *Using Apama with Software AG Designer*.
3. Copy the resultant directory onto each host that will run a correlator.
4. If necessary, edit the `environment.properties` file on each correlator host.
5. Ensure that the BigMemory and JMS servers are running.
6. On each correlator host, run the Ant deployment script to start the correlator.

If the project does not contain a distributed cache configuration, a local in-process MemoryStore will be used to store events. This is not shared or persistent, so only supports a single correlator deployment. If this correlator stops, it will drop all event history data. Apama recommends a BigMemory server and configuration for production use. See ["Deploying BigMemory Terracotta Server Array" on page 95](#) and ["Configuring BigMemory driver" on page 95](#).

Apama does not recommend running multiple correlators on a single machine. The assumption is that each correlator can use all of the CPU resources available. Also, running multiple correlators on one host does not provide any extra resilience. However, it is possible to run multiple correlators on a single machine. To do so:

1. Copy the exported deployment directory to separate directories on the correlator host machine.
2. Edit the `environment.properties` file to specify a different port number for each correlator and for each (if any) adapter in your project.

## Running queries on correlator clusters

---

The following topics describe how to run queries on correlator clusters.

### Deploying queries on multiple correlators

When using multiple correlators to deploy an Apama query application, it is the administrator's responsibility to keep the resources of the exported project up to date. If changes are made to a query, if queries are added or removed from a project, then all correlators should be updated to reflect the new state. It is possible to inject queries into a live running correlator, or delete queries from a correlator. Make sure that the injections and deletions are performed on all correlators in the cluster.

The queries runtime assumes that all members of a cluster:

- Share access to the same Distributed MemoryStore state - by using BigMemory Terracotta Server Array.

- Can connect freely between nodes.
- Run with clocks synchronized to within 1 second of each other. Apama recommends the use of the Network Time Protocol (NTP) to synchronize clocks.

The queries runtime will nominate a single member of the cluster as master, which will handle book keeping tasks such as garbage collecting nodes or handling failed cluster nodes.

If a correlator member of a cluster is using external clocking, then some functionality may not be available. The members will be able to share the same data, but an externally clocked node cannot be a master of other nodes and timers will not be failed over from an externally clocked node. In normal operation, external clocking should only be used for testing purposes on a single node (where failover and scalability is not required).

A production deployment of multiple nodes would not use external clocking for routine processing of events. Use the source timestamp feature (see *Using source timestamps of events* in *Developing Apama Applications*) if the events may be delayed or delivered out of order.

## Deploying BigMemory Terracotta Server Array

To deploy BigMemory, see the [BigMemory Max documentation](#).

For resilient operations, Apama recommends at least one backup on a separate host. You may want to consider using multiple stripes in order to improve performance. Ensure that the BigMemory server is accessible from all cluster members.

## Configuring BigMemory driver

---

### To configure the BigMemory driver

1. In Software AG Designer, add the Distributed MemoryStore adapter bundle to your Apama project.

In the **Distributed MemoryStore Configuration Wizard**, specify `ApamaQueriesStore` as the store name.

**Note:** If you specify a different store name or do not specify a name at all, an in-process only memory store will be used.

See also "Adding adapters to projects" in *Using Apama with Software AG Designer*.

2. Check that the cluster name is set correctly for the host/port pairs of all of the BigMemory Terracotta Server Array.
3. Set the **providerDir** property to the Terracotta installation directory.
4. Optionally, edit the on-heap and off-heap storage and other parameters as needed (see "BigMemory Max driver specific details" in *Developing Apama Applications*). The

`useCompareAndSwap` property should be left in its default `true` configuration for correct behavior of Apama queries.

## Using JMS to deliver events to queries running on a cluster

When running queries across multiple correlators in a cluster, as well as configuring all correlators to access the same BigMemory store, Apama recommends that all events are delivered into the cluster using a JMS queue. By using a JMS queue, each correlator will pull events from the JMS queue unless it has a full input queue (that is, it is behind on processing events) or has stopped running (e.g. shut down for maintenance or suffered a hardware failure). In either case, events will continue to be processed by other correlators in the cluster. Correlators can also be added to or removed from the cluster to scale the cluster capacity if desired. It is also possible to use per-correlator adapters for incoming events, but the adapters must co-ordinate so that every event is sent to only one correlator, and should one adapter/correlator pair fail, then other adapters process events that the failed node would have processed. Each event should only be delivered to one correlator, else multiple correlators will store the event in the shared cache, which can result in erroneous matches. Using JMS queues, this happens automatically, giving an 'elastic' system that can be scaled and continues running in the face of failure.

To run queries across multiple correlators in a cluster:

- Configure each correlator to access the same BigMemory store. This is a requirement.
- Use a JMS queue to deliver events into the cluster. This is a recommendation.

When the cluster uses a JMS queue, each correlator pulls events from the queue. If the input queue of one correlator in the cluster becomes full and it cannot pull events from the JMS queue the other correlators continue to do so and continue to process events. A correlator may stop pulling events because the correlator is behind on processing events or because it has stopped running, perhaps for maintenance or because of a hardware failure.

Using a JMS queue makes it easy to scale the cluster capacity by adding or removing correlators.

An alternative to using a JMS queue is to use an adapter for each correlator. For example, by having an IAF-based adapter connected to each correlator, it is possible to send messages to and from a query application without using JMS. A disadvantage of using per-correlator adapters is that the adapters must coordinate the following:

- Each event goes to only one correlator in the cluster. If an event goes to more than one correlator then multiple correlators store the same event in the shared cache. This can result in erroneous matches.
- Should one adapter/correlator pair fail then the other adapters process the events that the failed node would have processed.

Use of a JMS queue automatically ensures that an event goes to only one correlator and that all received events are processed. The result is an 'elastic' system that can be scaled and that continues to run even if a node fails.



Similar to using multiple contexts in a correlator, delivering events through JMS can result in events that occur close together in time being processed in an order that is different than the order in which they were created or sent to the JMS message bus. Queries do not support reliable message delivery. Consequently, if a correlator fails, perhaps because of a hardware failure, all events that had been received by the failed correlator but not yet written to BigMemory may be lost, and output events from queries can be lost. However, if the system is correctly scaled then the number of messages lost will be small.

Configure your JMS bus to have one or more queues, and configure a static JMS receiver connection. See "Getting started with simple correlator-integrated messaging for JMS" in *Connecting Apama Applications to External Components*. You will also need to provide mapping for all event types that flow into the queries. See "Mapping Apama events and JMS messages" also in *Connecting Apama Applications to External Components*.

The queries runtime ensures that after all queries have been injected into the correlator and started, they automatically start to receive events from JMS queues. There is no need to explicitly call `jms.onApplicationInitialized()` as described in "Using EPL to send and receive JMS messages" in *Connecting Apama Applications to External Components*.

For all applications that do not consist entirely of queries, for example, applications that contain additional EPL monitors or Java monitors, then it may be required to delay starting JMS until the application and queries are both ready to process events. The auto-starting of JMS behaviour of queries can be controlled by sending a `QueriesShouldNotAutoStartJMS()` event to the main context. This event can be routed by an application's `onload()` method. If this is done then a monitor in the main context should listen for a `QueriesStarted()` event and should wait until both the application and queries have started. The monitor can then call `jms.onApplicationInitialized()` directly. For example, the following monitor delays starting JMS until queries are started and a `StartMyApp()` event has been processed:

```
using com.apama.queries.QueriesShouldNotAutoStartJMS;
using com.apama.queries.QueriesStarted;
event StartMyApp {
}
monitor MyApp {
    import "JMSPlugin" as jms;
    action onload() {
        route QueriesShouldNotAutoStartJMS();
        on QueriesStarted() and StartMyApp() {
            jms.onApplicationInitialized();
        }
    }
}
```

## Mixing queries with monitors and scenarios

It is possible to have both monitors and queries in a project.

Events that are to be processed by queries should be sent to the `com.apama.queries` channel from monitors. Queries may send events to any channel, which EPL monitors may be subscribed to.

While queries will automatically scale and share state across a cluster, EPL monitors will not. Thus, be aware that a query may process subsequent events matching a pattern on different nodes. On different nodes, monitors with potentially different state will be executing. Similarly, the state of EPL monitors is not automatically stored in the distributed cache.

Both EPL monitors and Apama queries can make use of actions defined on events, subject to some limitations on the use of `spawn`, `die`, and event listeners. See "Restrictions in queries" in *Developing Apama Applications*.

## Handling node failure and failover

A node may stop processing events from time to time. This may be because it is stopped for planned maintenance, or the node failed in some way. In these cases:

- Events that have been delivered to the node but not yet processed will be lost. This will typically be a small window of events.
- Patterns that end with a `wait` operator may be dropped if they are being handled by a node that has stopped. For example, if a query contains the pattern `MyEvent:e - > wait(5.0)`, and the node that receives a matching `MyEvent` stops before 5 seconds are up, then the pattern will not be matched in this instance.
- If using JMS, then events continue to be delivered to and processed by other correlators in the cluster. The failed correlator will not hold up processing on other nodes. Other nodes continue processing events, including matching against events that the failed node had previously received (if they had been processed).
- Any clients connected to the failed correlator will need to re-connect to another correlator. The same set of parameterized query instances is kept in synchronization across the cluster. See ["Managing parameterized query instances" on page 98](#).

Similarly, nodes running a BigMemory Terracotta Server Array may fail. For this reason, BigMemory should be configured with sufficient backups to ensure no data is lost in this case. Consult the [Terracotta documentation](#).

## Managing parameterized query instances

---

When using parameterized queries, Apama recommends that you use one Scenario Service client at a time to manage parameterizations. Use of more than one client can result in undefined behavior if they both attempt to edit a parameterized instance concurrently. You can connect to any correlator in the cluster, and Apama will automatically synchronize the state of parameterized instances across the cluster. This assumes that the same query definitions have been injected into the correlators on all cluster nodes. If a node fails, you will need to connect to another correlator in the cluster.

## Creating new query instances by setting parameter values

Use Scenario Browser to set parameter values for a parameterized query and thus create new parameterized query instances, also referred to as parameterizations.

## Changing parameter values for queries that are running

Use Scenario Browser to change the parameter values for a running parameterized query instance, also referred to as a parameterization.

## Monitoring running queries

---

To help you monitor queries that are running on a given correlator, Apama provides data about active queries in DataViews. To display the information provided by these DataViews, you can create a dashboard in which an end user can:

- Monitor query runtime performance.
- Determine whether a query is behaving as intended. For example, you can see how incoming events are distributed across partitions. If you are expecting a particular send and match rate you can see if you are getting the results you expect.
- Ensure that the window size (the number of events in the window) is not too large. The expectation is that your application is designed so that partitioning keeps any given window size as small as possible.

The `Queries_Statistics_Sample` that is provided with Apama (located in `APAMA_HOME\samples\queries`) contains such a dashboard. It shows you how to build a dashboard that allows you to monitor the performance of running queries.

For information about exposing DataViews in dashboards, see *Building and Using Dashboards*.

A running query is either a non-parameterized query instance or a parameterization. For each running query, there is a DataView for each of its input event types. For example, if a query instance has two input event types then there are two DataViews that provide statistics for that query, one for each input event type.

Each DataView:

- Contains data about the activity during the last second of one running query and one of its input event types.
- Contains the fields described in the table below. The value contained in each field is an exponentially weighted moving average (EWMA).
- Is updated every second if the information has changed in the last second.

Statistical Field	Description
% Threads Active EWMA	<p>Apama uses multiple threads to process a given query. This is the percentage of those threads that were used within the last second to process the input event type that this DataView provides information for.</p> <p>While there is not a linear correlation, as this percentage goes down, the reliability of the rest of the statistics becomes weaker. This is because a smaller proportion of threads are contributing information.</p>
Avg. Inbound Event Rate/s EWMA	The average rate per second at which events of this type are being processed.
Avg. % of Successful Matches EWMA	The average percentage of the number of received events that cause a match.
No. Unique Keys Processed EWMA	The number of unique query partitions that were accessed for this event type within the past second.
Avg. Window Size/Key EWMA	The average window size (number of events that it contains) of each unique partition that was accessed within the past second.

The display name of these DataViews is **Correlator Query Statistics**.

After a non-parameterized query is injected into the correlator, Apama provides a DataView for each input event type and begins writing data to it. After a non-parameterized query is deleted, Apama no longer makes the DataViews for that query instance available.

For a parameterized query, after a parameterization is created then Apama adds new DataViews and begins populating them. When a parameterization is deleted then Apama no longer provides the DataViews that correspond to that parameterization. If the definition of a parameterized query is deleted then Apama no longer provides DataViews for any parameterizations of that query.

To help you monitor queries that are running across multiple correlators in a cluster, Apama also provides the same type of performance statistics provided for a given

correlator but where the underlying data has been aggregated across all the clustered correlators running those queries.

The display name of these DataViews is **Cluster-Wide Query Statistics**.

This means that for each query running on a correlator two types of monitoring data is provided:

- Statistics generated from data from only that correlator.
- Statistics generated from data aggregated across all correlators in the cluster running that same query.



# 5    Tuning Correlator Performance

---

■ Scaling up Apama .....	104
■ Partitioning strategies .....	105
■ Engine topologies .....	109
■ Event correlator pipelining .....	111

This section addresses how to scale up Apama to improve upon the performance of a single event correlator. It describes the Apama features you can use to send events to multiple event correlators to increase an application's capacity.

## Scaling up Apama

Apama provides services for real-time matching on streams of events against hundreds of different applications concurrently. This level of capacity is made possible by the advanced matching algorithms developed for Apama's event correlator component and the scalability features of the correlator platform.

Should it prove necessary, capacity can further be increased by using multiple event correlators on multiple hosts. To facilitate such multi-process deployments, Apama provides features to enable connecting components to pass events between them. It is recommended that each correlator is run on a separate host, to assist in the configuration of scaled-up topologies. However, it is possible to run multiple correlators on a single host. There are two methods of configuration:

- Using the configuration tools from the command line or Apama macros for Ant.
- Programmatically through a client programming API.

This guide describes both approaches, but first discusses different ways in which Apama can be distributed and what factors affect the choice of the distribution strategy.

**Note:** This topic focuses on scaling Apama for applications written in EPL. JMon has less scaling features as it does not support the use of multiple contexts. Java plug-ins can be used if invocation of Java code is required on multiple threads, either directly from EPL or by registering an event handler. See "Using Java plug-ins" in the "Developing Correlator Plug-ins" part of *Developing Apama Applications*. Knowledge of aspects of EPL is assumed, specifically monitors, spawning, listeners and channels. Definitions of these terms can be found in "Getting Started with Apama EPL" in *Developing Apama Applications*.

The core event processing and matching service offered by Apama is provided by one or more event correlator processes. In a simple deployment, Apama comprises a single event correlator connected directly to at least one input event feed and output event receiver. Although this arrangement is suitable for a wide variety of applications (the actual size depending on the hardware in use, networking, and other available resources), for some high-end applications it may be necessary to scale up Apama by deploying multiple event correlator processes on multiple hosts to partition the workload across several machines.

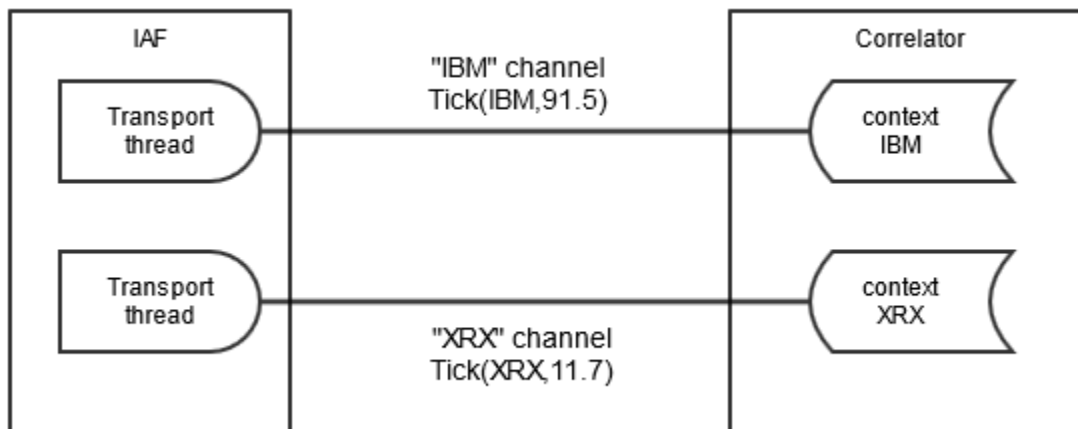


## Partitioning strategies

Using the patterns and tools described in this guide it is possible to configure the arrangement of multiple contexts within a single correlator or multiple event correlators within Apama (the engine topology). It is important to understand that the appropriate engine topology for an application is firmly dependent on the partitioning strategy to be employed. In turn, the partitioning strategy is determined by the nature of the application itself, in terms of the event rate that must be supported, the number of contexts, spawned monitors expected and the inter-dependencies between monitors and events. The following examples illustrate this.

The `stockwatch` sample application (in the `samples\monitorscript` folder of your Apama installation directory) monitors for changes in the values of named stocks and emits an event should a stock of interest fall below a certain value. The stocks to watch for and the prices on which to notify are set up by initialization events, which cause monitors that contain the relevant details to be spawned. In this example, the need for partitioning arises from a very high event rate (perhaps hundreds of thousands of stock ticks per second), which is too high a rate for a single context to serially process.

A suitable partitioning scheme here might be to split the event stream in the adapter, such that different event streams are sent on different channels. The illustration below shows how this can be accomplished:



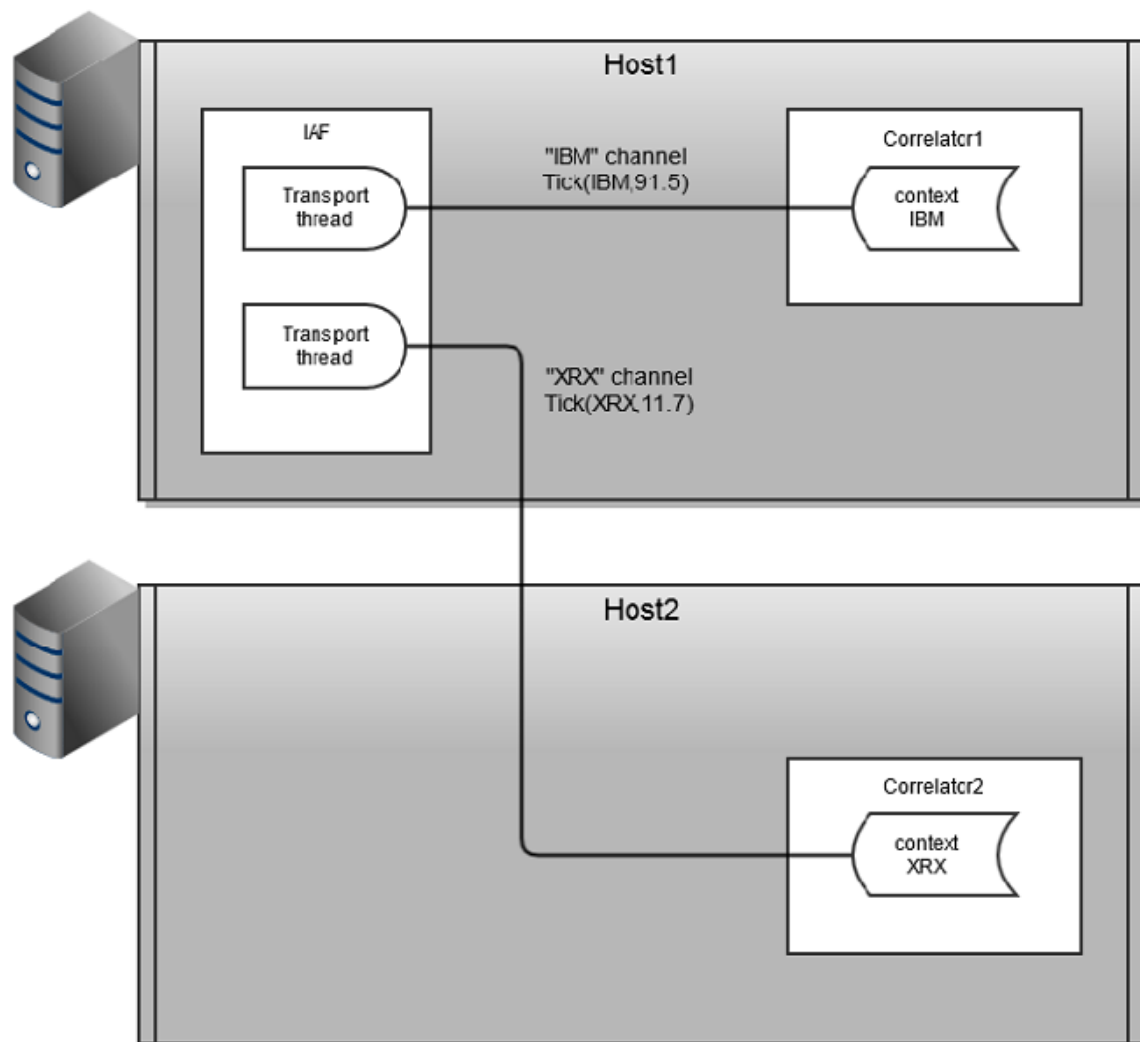
This diagram shows an adapter sending events to different channels based on the symbol of the stock tick. The adapter transport configuration file would specify a `transportChannel` attribute for the stock event that named a field in the `NormalisedEvent` that specified the stock symbol. Either a thread per symbol or a single thread (which could become a bottleneck) managed by the transport, depending on what the system the transport is connecting to allows, is used to send `NormalisedEvents` to the semantic mapper to be processed. The IAF thus sends the events on the channel in the stock symbol value in the `NormalisedEvent`.

In this example, the stock symbol is either `IBM` or `XRX`. The IAF will send events to all sinks (typically one) that are specified in the IAF's configuration file. In the correlator,

all monitors interested in events for a given symbol would need to set up listeners in a context where a monitor has subscribed to that symbol. To achieve good scaling, the application is arranged so that each context is subscribed to only one symbol. For the `stockwatch` application, a separate context per symbol would be created, and the `stockwatch` monitor spawns a new monitor instance to each context. In each context, the monitor instance would execute `monitor.subscribe(stockSymbol)`; where `stockSymbol` would have the value "IBM" or "XRX" corresponding to the stock symbol it is interested in. This application will scale well, as each event stream (for the different stock symbols) can run in parallel on the same host; this is referred to as scale-up.

Listeners in each context would listen for events matching a pattern, such as `on all Tick(symbol="IBM", price < 90.0)`.

If the number of stock symbols is very large and the amount of processing for each stock symbol is large, then it may be required to run correlators on more than one host to use more hardware resources than are available in a single machine. This is referred to scale-out. To achieve scale-out, connections per channel need to be made between the Apama components using the `engine_connect` tool (or the equivalent call from Ant macros or the client API). The `engine_connect` tool can connect any two Apama components, either correlator to correlator, or IAF to correlator. For best scaling, multiple connections are required between components, which `engine_connect` provides in the parallel mode. The following image shows a scaled out configuration.



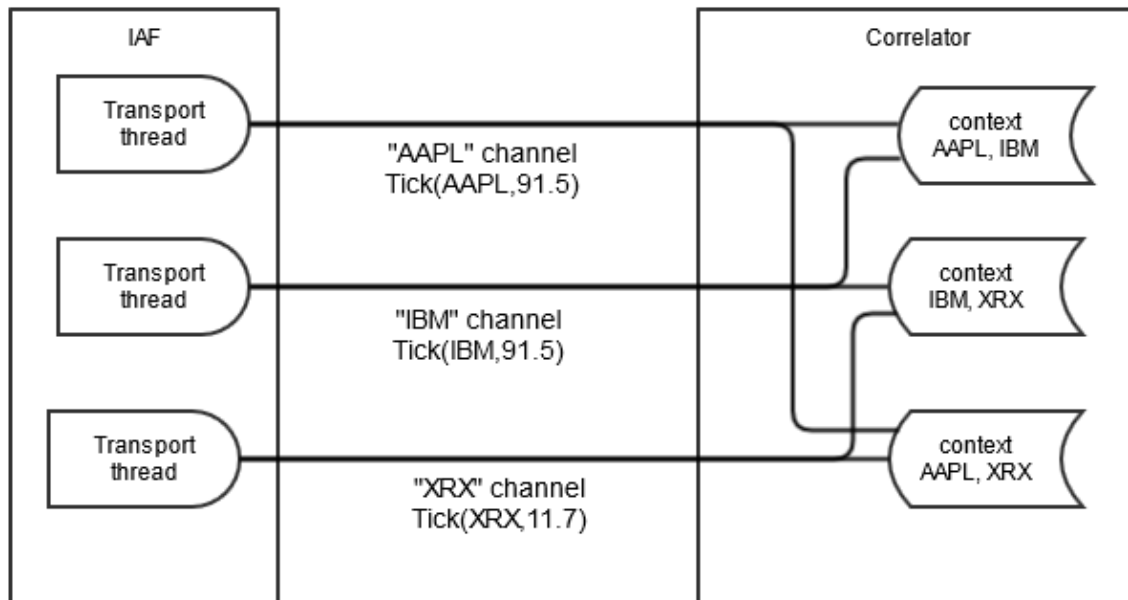
This configuration allows many contexts to run on two hosts and requires use of `engine_connect` to set up the topology.

Now consider a portfolio monitoring application that monitors portfolios of stocks, emitting an event whenever the value of a portfolio (calculated as the sum of stock price \* volume held) changes by a percentage. A single spawned monitor manages each portfolio and any stock can be added to/removed from a portfolio at any time by sending suitable events.

This application potentially calls for significant processing with each stock tick, as values of all portfolios containing that stock must be re-calculated. If the number of portfolios being monitored grows very large, it may not be possible for a single context to perform the necessary recalculations for each stock tick, thus requiring the application to be partitioned across multiple contexts.

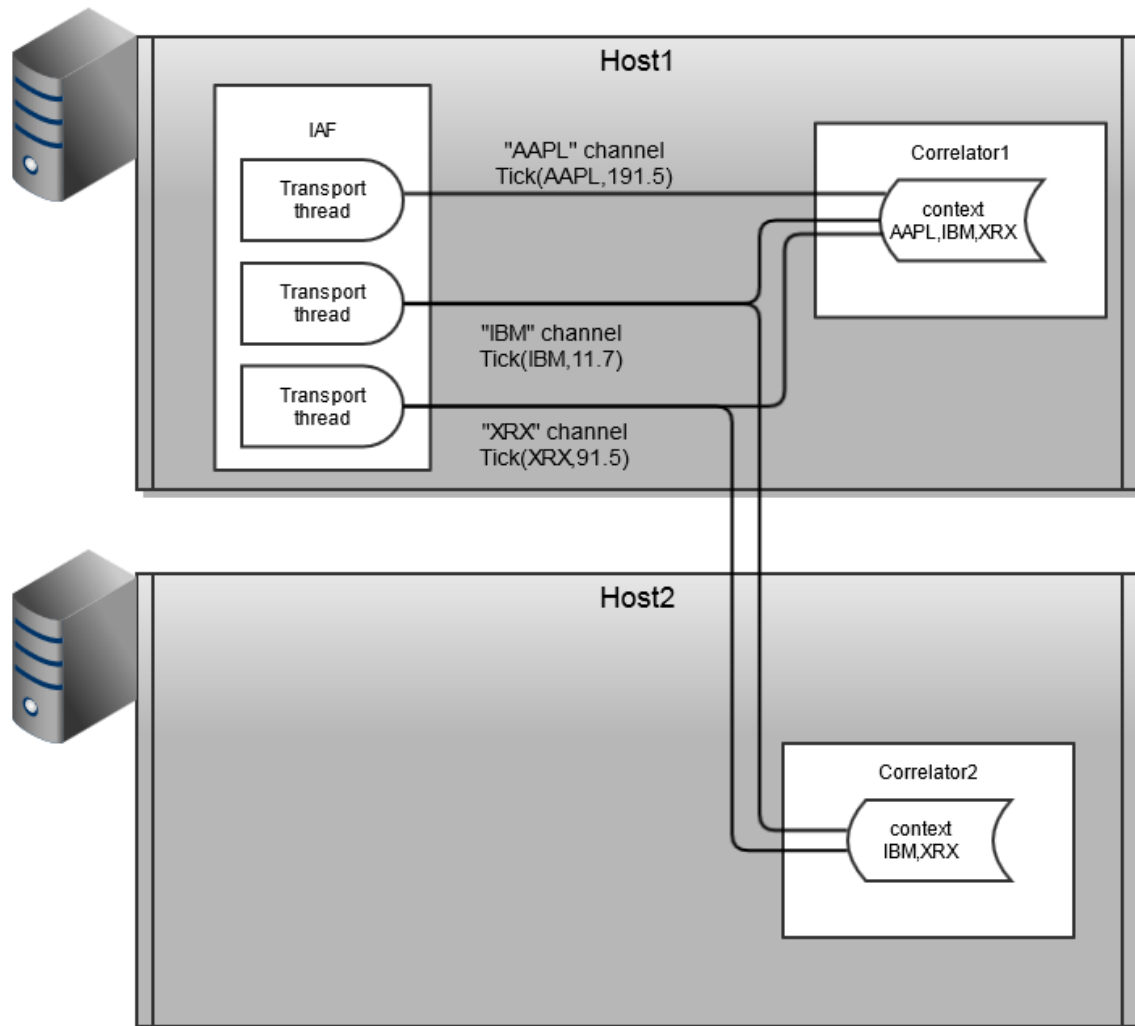
Unlike the stockwatch application, it is not possible to achieve scaling to larger numbers of portfolios by splitting the event stream. Each portfolio can contain multiple stocks, and stocks can be dynamically added and removed, thus one event may be required by

multiple contexts. In this case, a suitable partitioning scheme is to partition the monitor instances across contexts (as with `stockwatch`) but to duplicate as well as split the event stream to each event correlator. The following images shows the partitioning strategy for the portfolio monitoring application.



Again, each monitor instance is spawned to a new context and subscribes to the channels (stock symbols in this application) that it requires data for. Note that while the previous example would scale very well, this will not scale as well. In particular, if one monitor instance requires data from all or the majority of the channels, then it can become a bottleneck. However, there may be multiple such monitor instances running in parallel if they are running in separate contexts.

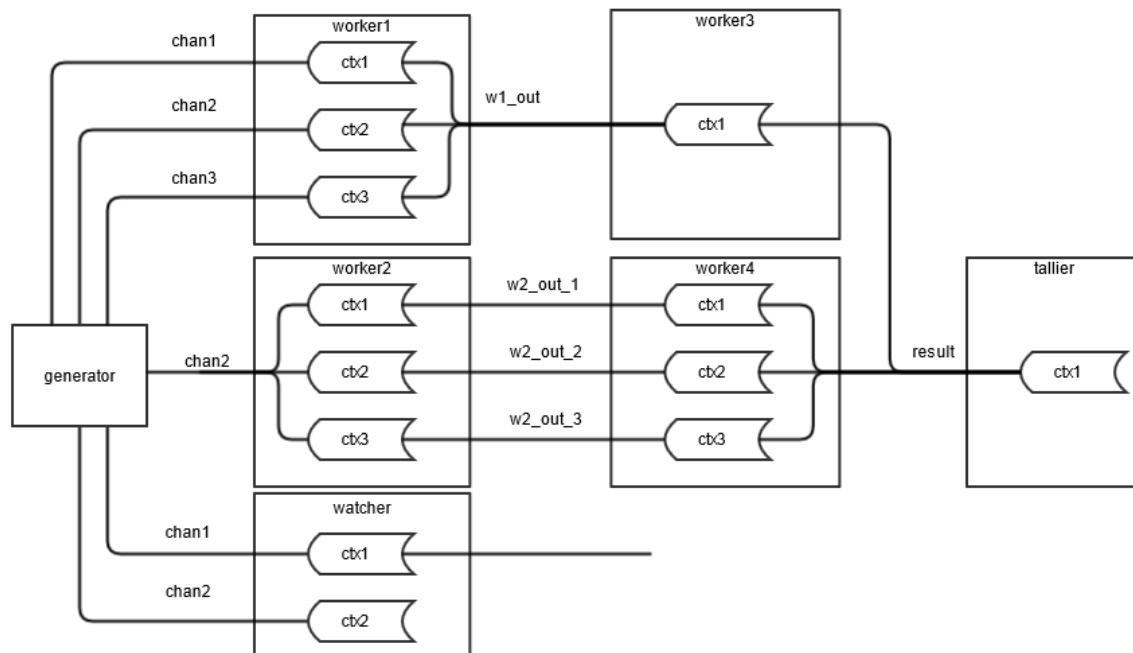
Similar to the `stockwatch` application, the portfolio monitoring application may require scale-out across multiple hosts, as shown below.



In summary, the partitioning strategy can be thought of as a formula for splitting and duplicating monitors and/or events between event correlators while preserving the correct behavior of the application. In some circumstances, it may be necessary to re-write monitors that work correctly on a single correlator to allow them to be partitioned across correlators, as the following section describes.

## Engine topologies

Once the partitioning strategy has been defined, in terms of which events and monitors go to which correlators, it is necessary to translate this into an engine topology. This is achieved by connecting source and target event correlators on separate channels, such that events sent by a source correlator on a specific channel find their way to the correct contexts in the target correlator. A set of two or more event correlators connected in this way is known as a correlator pipeline, as shown in the following image. This figure represents an example topology for a high-end application – the majority of applications use a single correlator only, or have far simpler topologies.



In this image, an event correlator can perform the function of each of the 7 nodes (generator, worker, watcher, tallier). Each target correlator performs some processing before passing the results to a second worker correlator (worker3, worker4) in the form of events, sent on the channels as marked on the diagram. `tallier` collates the results from these correlators for forwarding to any registered receivers. A final correlator, `watcher`, monitors the events emitted by `generator` on `chan1` and `chan2` and emits events (possibly containing status information or statistical analysis of the incoming event stream) to any registered receivers.

To deploy an application on a topology like that shown above requires separating the processing performed into a number of self-contained chunks. In the previous figure, it is assumed that the core processing can be serialized into three chunks, with the first two chunks split across two correlators each (worker1/2 and worker3/4 respectively) and the third chunk residing on a single correlator (`tallier`). Intermediate results from each stage of processing are passed to the next stage as sent events, which contexts in the connected correlators receive by subscribing to the appropriate channels.

To realize this application structure requires coding each chunk of processing as one or more separate monitors, which send intermediate results as an event of known type on a pre-determined channel. These monitors can then be loaded onto the appropriate correlator. This may require an existing application that grows beyond the capacity of a single event correlator, to be re-written as a number of (smaller) monitors to allow partitioning of the application processing into separate chunks in the manner described above.

## Event correlator pipelining

To implement engine topologies comprising multiple event correlators requires a method of connecting correlators in pipelined configurations. This can be achieved in the following ways:

- Directly using the `engine_connect` tool. See ["Configuring pipelining with engine\\_connect" on page 111](#).
- Indirectly using Software AG's Universal Messaging (UM) message bus. For complex deployments where parts of the application may be moved between Apama correlators, this is likely to be the best alternative. When using UM each correlator connects to the same UM realm. See "Using Universal Messaging in Apama Applications" in *Connecting Apama Applications to External Components*.
- Programmatically via the client API, see ["Configuring pipelining through the client API" on page 118](#).
- Using a custom launch configuration in the Software AG Designer. See "Connecting correlators" in *Using Apama with Software AG Designer*.

## Configuring pipelining with engine\_connect

The `engine_connect` tool allows direct connection of running correlator instances. The executable for this tool is located in the `bin` directory of the Apama installation. Running the tool in the Apama Command Prompt (see ["Setting up the environment using the Apama Command Prompt" on page 26](#)) ensures that the environment variables are set correctly.

### Synopsis

To configure pipelining, run the following command:

```
engine_connect [ options ]
```

When you run this command with the `-h` option, the usage message for this command is shown.

### Description

`engine_connect` connects a source correlator (the sender) to a target correlator (the receiver). The target correlator will receive events from the specified channel(s) of the source correlator. Source and target correlators must already be running.

Alternatively, if you specify the `-f` option, `engine_connect` reads connection information from the specified file and sets up each connection found therein (see ["Configuring pipelining through the client API" on page 118](#) for details of the file format). The `engine_connect` tool expects the specified file to be in the local character set. If the configuration file is in UTF-8, specify the `-u` option in addition to the `-f`

option. If the filename provided to `-f` is a hyphen (`-`), then connection information is read from the standard input device (`stdin`) until end-of-file.

The connection between the source and target correlators is persistent. When one of the correlators stops running, then when that correlator restarts it automatically reconnects with the other correlator.

The tool is silent by default unless an error occurs. For verbose progress information, use the `-v` option.

## Options

The `engine_connect` tool takes the following options:

Option	Description
<code>-h   --help</code>	Displays usage information.
<code>-sn host   --sourcehost host</code>	Name of the host on which the source (event sending) correlator is running. The default is <code>localhost</code> . However, you can use the default or specify <code>localhost</code> only when the source correlator and the target correlator are running on the same host. In all other situations, you must specify the public IP address or the name of the host. This ensures that the host of the target correlator can resolve the name/address of the source correlator host. Non-ASCII characters are not allowed in host names.
<code>-sp port   --sourceport port</code>	Port on which the source (event sending) correlator is listening. The default is <code>15903</code> .
<code>-tn host   --targethost host</code>	Name of the host on which the target (event receiving) correlator is running. The default is <code>localhost</code> . However, you can use the default or specify <code>localhost</code> only when the source correlator and the target correlator are running on the same host. In all other situations, you must specify the public IP address or the name of the host. This ensures that the host of the source correlator can resolve the name/address of the target correlator host. Non-ASCII characters are not allowed in host names.
<code>-tp port   --targetport port</code>	Port on which the target (event receiving) correlator is listening. The default is <code>15903</code> .



Option	Description
<code>-c channel   --channel channel</code>	<p>Named channel on which to send/receive events. You can specify the <code>-c</code> option multiple times to send/receive events on multiple channels. You must specify the <code>-c</code> option at least once for each sender/target pair. Until you do, no events emitted by the sender correlator are received by the target correlator. An event is discarded if it is sent on a channel for which you did not specify the <code>-c</code> option.</p>
<code>-m mode   --mode mode</code>	<p>Indicates whether there is one connection (<code>-m legacy</code>) between the sender and target correlators or one connection for each specified channel (<code>-m parallel</code>).</p> <p>The default behavior is that there is one connection between the sender and target correlators. The tool uses the same connection for every channel. Events sent on any channel are delivered to the default channel in the target correlator and all events are delivered in order. You can specify default behavior by specifying <code>-m legacy</code> or <code>--mode legacy</code>.</p> <p>To create a connection for each specified channel, specify <code>-m parallel</code> or <code>--mode parallel</code>. Events sent on a named channel are delivered to the same named channel in the target correlator. Events sent on the same channel are delivered in order. Events sent on different channels may be re-ordered.</p> <p>You also specify the <code>-m</code> option when you specify the <code>-x</code> option to disconnect. If you are using a separate connection for each channel, you should specify <code>-m parallel</code> when you specify the <code>-x</code> option. If you are using one connection for all channels, you should specify <code>-m legacy</code> when you specify the <code>-x</code> option.</p> <p>See also <a href="#">"Avoid mixing connection modes" on page 116</a>.</p>
<code>-x   --disconnect</code>	<p>When you specify the <code>-x</code> option, the behavior depends on whether you also specify the <code>-c</code> option.</p> <p>If you specify the <code>-x</code> option and you do not also specify the <code>-c channel</code> option, then the source correlator stops sending events to the target correlator. Each connection between the source correlator and the target correlator is terminated.</p>

Option	Description
	<p>If you specify the <code>-x</code> option and the <code>-c channel</code> option and the tool is using one connection for each channel, then the source correlator terminates only the connection(s) it was using for the channel(s) you specify. Any other connections being used for other channels continue to be used. You can specify the <code>-x</code> option with one or more instances of the <code>-c channel</code> option. Remember to also specify <code>-m parallel</code>.</p> <p>If you specify the <code>-x</code> option and the <code>-c channel</code> option and the tool is using one connection for all channels, then the source correlator stops sending events on only the channel(s) you specify. The source correlator continues to send events on any other channels it was already sending events on. If there are no other channels, then the source correlator no longer sends events to the target correlator. However, the connection between the two correlators remains in place. Remember to also specify <code>-m legacy</code>.</p>
<code>-s   --qdisconnect</code>	Disconnect if slow (only takes effect on the first connection).
<code>-f file   --filename file</code>	Read connection information from the named file. If this option is specified, the options <code>-sn</code> , <code>-sp</code> , <code>-tn</code> , <code>-tp</code> and <code>-c</code> are all ignored. This file must be in the local character set or in UTF-8 format. If it is UTF-8, specify the <code>-u</code> option in addition to this option.
<code>-u   --utf8</code>	Indicates that the connection information file is in UTF-8.
<code>-v   --verbose</code>	Requests verbose output during <code>engine_connect</code> execution.
<code>-V   --version</code>	Displays version information for the <code>engine_connect</code> tool.

### Exit status

The `engine_connect` tool returns the following exit values:

Value	Description
0	All connections were established successfully.
1	One or more source correlators could not be contacted.
2	One or more target correlators could not be contacted.
3	A problem occurred establishing the connection; request invalid.
4	Target correlator failed to contact the source.
5	Some other error occurred.

### Comparison of legacy and parallel connection modes

Legacy connection mode	Parallel connection mode
0 or 1 connection between two correlators.	Any number of connections between correlators.
Events sent on different channels are delivered in the order in which they are sent.	Events sent on different channels may be delivered in a different order from the order in which they were sent.
Sending an event to a named channel delivers the event to the default channel.	Sending an event to a named channel delivers it to only that channel.
Unlike Universal Messaging for passing events between correlators.	Similar to Universal Messaging for passing events between correlators.
Same behavior as releases earlier than Apama 5.2.	New behavior starting with Apama 5.2.
Universal Messaging has no mechanism for enforcing ordering among events sent on different channels. However, Universal Messaging is the better alternative when you want to use a large number of channels to send events between components. Without Universal Messaging, the use of two TCP connections with threads on both ends of the connection might reach the limit of how many channels can have dedicated connections.	

### Avoid mixing connection modes

Successive command lines that specify the same source/target hosts/ports build on each other. While this makes it possible to mix the legacy and parallel connection modes, you should avoid doing that. Mixing connection modes can cause an event to be delivered twice to the same channel. For example:

```
engine_connect -tp 15902 -sp 15903 -c channelA -c channelB
engine_connect -tp 15902 -sp 15903 -c channelA -c channelC -m legacy
```

The result of the first command is that there is one (legacy) connection for sending/receiving events on `channelA` and `channelB`. The result of the second command is that there is a dedicated connection for sending/receiving events on `channelA` and a dedicated connection for sending/receiving events on `channelC`. Events sent on `channelA` would be delivered twice: once on the legacy connection and once on the dedicated connection.

### Examples

Because you can specify command lines that build on each other, you could set up a connection and add named channels later. You can also unsubscribe the channels you have added so that no events are sent or received. The connection remains and you can re-add channels at a later time. However, until you specify the `-c` option for a given connection, no events emitted by the source correlator are received by the target correlator. Consider the following command line:

```
engine_connect -sn host1 -sp 15903 -tn host2 -tp 15904
```

The correlators on `host1` and `host2` are connected but no channels have been subscribed and therefore no events are sent/received. To send and receive events, specify the `-c` option as in the following command line:

```
engine_connect -sn host1 -sp 15903 -tn host2 -tp 15904 -c CHAN1 -c CHAN2
```

Now the connected correlators can use `CHAN1` and `CHAN2` to send/receive events. To add another channel, execute this command:

```
engine_connect -sn host1 -sp 15903 -tn host2 -tp 15904 -c CHAN3
```

The correlators are now using `CHAN1`, `CHAN2`, and `CHAN3` to send/receive events. To stop using `CHAN2`, execute the following command. The correlators continue to use `CHAN1` and `CHAN3`.

```
engine_connect -sn host1 -sp 15903 -tn host2 -tp 15904 -x -c CHAN2
```

To stop sending and receiving events, execute the following command. Note that the correlators remain connected until one of them stops. There is no penalty for this connection.

```
engine_connect -sn host1 -sp 15903 -tn host2 -tp 15904 -x
```

In this example, the following command is equivalent to the previous command.

```
engine_connect -sn host1 -sp 15903 -tn host2 -tp 15904 -x -c CHAN1 -c CHAN3
```

## Connection configuration file

`engine_connect` can take connection information from a file for connecting and disconnecting event correlators. A sample of such a configuration file is shown below, which defines the topology shown in ["Engine topologies" on page 109](#).

```
# comments are allowed prefixed by a '#' - the rest of the line
# is ignored
generator:dopey.apama.com:1234
worker1:sleepy.apama.com:1234:generator{chan1,chan2,chan3}
worker2:grumpy.apama.com:1234:generator{chan2}
worker3:sneezy.apama.com:1234:worker1{w1_out}
worker4:bashful.apama.com:1234:worker2{w2_out_1,w2_out_2,w2_out_3}
tallier:happy.apama.com:1234:worker3{result},worker4{result}
watcher:doc.apama.com:1234:generator{chan1,chan2}
```

### Connection configuration file format

Each entry in the configuration file specifies connection information for a single correlator in the deployment. Entries can be specified in any order. The general format of an entry is:

```
correlator_name [:host] [:port] [:connection [,connection ...]]
```

where `<connection>` is defined as:

```
correlator_name [ {channel_name [,channel_name ...]}]
```

*correlator\_name* is a symbolic identifier for a correlator, used to identify source correlators in target correlator connection information. It can consist of any combination of characters other than whitespace, colon, comma or open/close brace characters, which are reserved as separators. *host* and *port* identify the specific correlator this entry applies to. They can be omitted, in which case the defaults of `localhost` and `15903` are used respectively.

Following this information are details of all connections to source correlators for the current (target) entry. This information is omitted if no correlators sit 'upstream' of the current entry (as with the correlator `generator`, above). If there are multiple upstream source correlators, each name should be separated by a comma (as with `tallier`, above, which takes events from `worker3` and `worker4`).

For each connection it is possible to specify the channel(s) on which the target correlator will listen. If no channels are specified the target correlator will register to receive all events emitted by the source correlator regardless of channel (as with correlators `worker3` and `worker4` which register for all events from `worker1` and `worker2` respectively). One can specify specific channel names by enclosing them in braces and separating multiple channels by commas (as with `watcher` which registers with `generator` for all events on channels `chan1` and `chan2`).

In effect, the configuration file is a convenient way of grouping several calls to `engine_connect`. For example to set up the connections for the correlator `tallier` would require two commands using `engine_connect`:

```
>engine_connect -m parallel -sn sneezy.apama.com -sp 1234 -tn happy.apama.com
    -tp 1234 -c result
```

```
>engine_connect -m parallel -sn bashful.apama.com -sp 1234 -tn happy.apama.com
      -tp 1234 -c result
```

### Errors in the configuration file

The configuration file can be used to both establish and remove connections in a multi-correlator engine topology. For example, assuming the above file is saved as `topology.dat`, the following commands will first set up then tear down all the connections specified therein:

```
>engine_connect -m parallel -f topology.dat
>engine_connect -m parallel -x -f topology.dat
```

In each of these cases, `engine_connect` will exit with non-zero exit status on the first error it detects in the configuration file. An error message will be printed to standard error (`stderr`).

### Re-playing the configuration file

The behavior of `engine_connect` without the `-x` option is additive. This means that successive calls to `engine_connect` will attempt to add the channels specified to any existing connection between the source and target correlator(s). For example, with reference to the configuration file above, these commands:

```
>engine_connect -m parallel -sn dopey.apama.com -sp 1234 -tn sleepy.apama.com
      -tp 1234 -c foo
>engine_connect -m parallel -f topology.dat
```

will first add a connection from correlator `generator` to worker1 on channel `foo`, then (from the configuration file) extend that connection so that worker1 also receives all events from `generator` emitted on channel `chan1`.

Once a connection is set up between two correlators on a channel, any further attempt to set up that connection on the same channel will have no effect. It is therefore possible to re-play the configuration file by invoking `engine_connect` without creating duplicate connections. This can be useful if there is an error in the configuration file signaled when `engine_connect` is called, as the error can be fixed and `engine_connect` re-run without requiring removal of connections that were successfully set up by the first call to `engine_connect`.

## Configuring pipelining through the client API

Apama provides client software development kits (SDKs) that can be used to interface with a running event correlator or group of event correlators. You can use `attachAsConsumerOfEngine` of the `EngineClient` API in Java and .NET (or `attachAsConsumerOfEngine` of the lower-level `EngineManagement` API in Java, .NET, C++ and C). For more information, see "Developing Custom Clients" in *Connecting Apama Applications to External Components*.

## Event partitioning

Using `engine_connect` or the Apama client library, it is possible to create topologies of correlators across which an application's monitors can be partitioned. Use the

`engine_inject` tool described in ["Injecting code into a correlator" on page 150](#), or by means of the relevant functions of the client library, to load the relevant monitors directly on to the appropriate correlators, specifying the host and port for each correlator.

This scheme is suitable for most applications, as monitors can be loaded once when Apama is brought online. For some applications, however, there is a requirement for a dynamic routing mechanism that (depending on the requirements of the application) continually splits and/or duplicates the incoming event stream and sends it to two or more correlators. Use the IAF `transportChannel` attribute to specify the channel an event is sent to, and connect that channel to the appropriate correlators.





# 6 Managing and Monitoring over REST

---

- Generic Management ..... 125
- Correlator Management ..... 126
- IAF Management ..... 127
- Dashboard Management ..... 127

Apama provides a Representational State Transfer (REST) HTTP API with which you can monitor Apama components. The monitoring capabilities are available to third-party managing and monitoring tools or to any application that supports sending and receiving XML documents, or receiving JSON documents, over the HTTP protocol.

Apama components expose several URIs which can be used to either monitor or manage different parts of the system. Some are exposed by most Apama components. These are the generic management URIs. Some are exposed only by specific types of components. For example, a correlator running on the default port of 15903 will expose a URI at `http://localhost:15903/correlator/status`. If an HTTP `GET` is issued against the URI, the correlator will return a document with the current status of the correlator. The format of this document is depicted by the header set in the request, that is, `application/xml` for XML and `application/json` for JSON.

Most URIs are purely for informational purposes and will only respond to HTTP `GET` requests, and interacting with them will not change the state of the component. However, some URIs allow the state of the correlator to be modified. This is currently supported for XML only. They will respond to one or more of the other HTTP methods. For example, the `/logLevel` URI will accept an HTTP `PUT` request containing an XML document describing what the log level of the component should be set to. Note that in this case, it will also accept a `GET` request which will report the current log level of the component, in keeping with REST principles.

All requests and responses over these interfaces have the same, simple elements:

- In XML, these elements are:

```
prop
map
list
```

All elements have a `name` attribute. The `prop` element simply represents a name-value pair with the name contained in the `name` attribute and the value being the content of the element. The `map` element is an unordered list of named elements which might be any of the three sets of elements, though it is quite typically simply a map of `prop` elements. See the `/info` URI as an example. The `list` is very similar to the `map` element except that here the order is typically regarded as significant. All responses from these URIs have a top-level element with the name `apama-response` and similar all requests which are sent to these URIs should have a top-level element with the name `apama-request`. If there is an error, then a response called `apama-exception` will be returned.

- In JSON, these elements are:

```
map {}
list []
```

All elements have a name-value pair. Name and value are separated by a colon (`:`) with the name to the left and the value to the right of the colon. The `map` element, which is represented by curly brackets, is an unordered list of named elements which might be any of `map` or `list` elements. See the `/info` URI as an example. The `list` element, which is represented by square brackets, is very similar to

the `map` element except that here the order is typically regarded as significant. If there is an error, then a response with the error message is returned, for example, `{"apamaErrorMessage":"Not found"}`.

For both formats, the `/connections` URL is a good example of all these elements being used together:

- In XML, the top-level element is a `map` which has two children, both `list` elements, called `senders` and `receivers`. Each list contains a `map` element for each sender and receiver. Each sender or receiver has a set of `prop` elements.
- In JSON, the top-level element is a `map {}` which has two children, both `list []` elements, called `senders` and `receivers`. Each list contains a `map {}` element for each sender and receiver. Each sender or receiver has a set of name-value pairs.

Method	URI	Supported Format	Description
GET	<code>/info</code>	XML, JSON	Summary information about the component including its name, version, etc.
GET	<code>/ping</code>	XML	Check if the component is running.
GET	<code>/deepPing</code>	XML	Check if the component is running.
GET	<code>/logLevel</code>	XML, JSON	Display the current log level of the component.
PUT	<code>/logLevel</code>	XML	Issues a request to change the log level of the component. Not supported for dashboard servers.
GET	<code>/connections</code>	XML, JSON	Display the connections to the component. For dashboard servers, this is always empty.
GET	<code>/info/argv</code>	XML	Display the arguments that were specified when the component was started.

Method	URI	Supported Format	Description
GET	/info/envp	XML	Display the names and values of the environment variables in use.
GET	/info/sysprop	XML, JSON	Dashboard servers only. Display the names and values of the Java system properties of the component.
GET	/info/categories	XML	Display the categories available; for example <code>argv</code> and <code>envp</code> .
GET	/correlator/status	XML, JSON	Display the runtime status of a running correlator and the user-defined status values.
GET	/correlator/info	XML, JSON	Display information about the state of a running event correlator.
GET	/correlator/types	XML, JSON	Display all the types currently known to the correlator.
GET	/correlator/appLogging	XML, JSON	Display information around the application logging at different levels.
GET	/iaf/status	XML, JSON	Summary information about the IAF component.

Two examples are provided below, one for JSON and another for XML. Each example shows only a possible hierarchy of a response. To get the actual format of the response for each request, it is recommended that you actually make the request.

Example for JSON:

```
{
  "Key1": [
    {"Key1.1.1": "Value1.1.1", "Key1.1.2": "Value1.1.2"},
    {"Key1.2.1": "Value1.2.1", "Key1.2.2": "Value1.2.2"}
  ],
}
```

```

"Key2": [],
"Key3":
[
  { "Key3.1.1": "Value3.1.1", "Key3.1.2": [] }
],
"Key4": []
}

```

### Example for XML:

```

<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="/resources/transform.xslt"?>
<map name="apama-response">
  <list name="Key1">
    <map name="Key1.1">
      <prop name="Key1.1.1">Value1.1.1</prop>
      <prop name="Key1.1.2">Value1.1.2</prop>
    </map>
    <map name="Key1.2">
      <prop name="Key1.2.1">Value1.2.1</prop>
      <prop name="Key1.2.2">Value1.2.2</prop>
    </map>
  </list>
  <list name="Key2"/>
  <list name="Key3">
    <map name="Key3.1">
      <prop name="Key3.1.1">Value3.1.1</prop>
      <list name="Key3.1.2"/>
    </map>
  </list>
  <list name="Key4"/>
</map>

```

## Generic Management

The Apama REST API `GET` methods return information about correlators, sentinel agents and IAFs, and the `PUT /logLevel` method changes the specified log level. The `GET` methods also return information on the dashboard data servers and display servers, with the restrictions mentioned below.

### GET /info

This method returns summary information about the component including its name, version, etc. This is analogous to the data that can be retrieved with the Apama `engine_management` tool, for example, the `hostname` field is exactly what `engine_management --gethostname` gives you.

### GET /ping and GET /deepPing

Checks if the component is still running. If the component is running the client receives an empty response. A failure is a timeout waiting for the response.

### GET /logLevel

This method displays the log level of a component.

**PUT /logLevel**

This methods sets the log level of a component. Not supported for dashboard servers.

**GET /connections**

Gets the incoming and outgoing messaging connections to the given component, along with the channels subscribed to and whether or not the receivers are slow. For dashboard servers, this is always empty.

**GET /info/argv**

This method returns the arguments used when starting the component as a list of name-value pairs.

**GET /info/envp**

This method returns a list the environment variables in use as a list of name-value pairs.

**GET /info/sysprop**

Dashboard servers only. This method returns the names and values of the Java system properties of the component.

**GET /info/categories**

This method returns the names of the categories for which you can get general information.

---

## Correlator Management

---

The Apama REST API provides URIs to use with the `GET` method in order to return information about running Apama correlators. This information includes data such as the number of listeners, monitors, and contexts in the correlator, the number and types of events, and the number of JMon applications. For details about the type of information returned by these methods, see ["Watching correlator runtime status" on page 164](#) and ["Inspecting correlator state" on page 170](#).

**GET /correlator/status**

This is analogous to the information reported by the Apama `engine_watch` tool.

**GET /correlator/info**

This is analogous to the information reported by the Apama `engine_inspect` tool.

## IAF Management

---

The Apama REST API provides a URI to use with the `GET` method in order to return information about adapters running in the Apama Integration Adapter Framework (IAF). This information includes data such as the number of codecs and transports in use and the number of events sent and received.

### **GET /iaf/status**

This returns the same information as the output of the Apama `iaf_watch` tool.

## Dashboard Management

---

The Apama REST API does not provide a dashboard-specific management interface. You can manage the information about the dashboard data server and display server components either as described in "[Generic Management](#)" on page 125 or using the `dashboard_management` tool (see "Managing and stopping the Data Server and Display Server" in *Building and Using Dashboards*).

The `component_management` tool can also be used for many of these tasks. See "[Shutting down and managing components](#)" on page 172.





# 7

## Correlator Utilities Reference

■ Starting the event correlator .....	130
■ Injecting code into a correlator .....	150
■ Deleting code from a correlator .....	153
■ Packaging EPL and Java code .....	156
■ Sending events to correlators .....	158
■ Receiving events from correlators .....	162
■ Watching correlator runtime status .....	164
■ Inspecting correlator state .....	170
■ Shutting down and managing components .....	172
■ Using the command-line debugger .....	199
■ Generating code coverage information about EPL files .....	209
■ Replaying an input log to diagnose problems .....	216
■ Event file format .....	219
■ Using the data player command-line interface .....	223
■ Using the Apama component extended configuration file .....	225

The Apama event correlator is at the heart of Apama applications. The correlator is Apama's core event processing and correlation engine. This section provides information and instructions for using command-line tools to monitor and manage event correlators.

For information about EPL, event definitions, monitors, namespaces and packages, see "Getting Started with Apama EPL" in *Developing Apama Applications*.

## Starting the event correlator

The `correlator` tool starts the event correlator. The executable for this tool is located in the `bin` directory of the Apama installation. Running the tool in the Apama Command Prompt (see ["Setting up the environment using the Apama Command Prompt" on page 26](#)) ensures that the environment variables are set correctly.

### Synopsis

To start the event correlator, run the following command:

```
correlator [ options ]
```

When you run this command with the `-h` option, the usage message for this command is shown.

### Description

By default, the `correlator` tool starts an event correlator process on the current host, and configures it to listen on port 15903 for monitoring and management requests.

On start-up, the executable displays the current version number and configuration settings.

To terminate an event correlator process, press Ctrl+C in the window in which it was started. Alternatively, you can issue the `engine_management` command with the `--shutdown` option. See ["Shutting down and managing components" on page 172](#). Regardless of which technique you use to terminate the correlator, Apama first tries to shut down the correlator cleanly. If this is not possible, for example, perhaps because of a monitor in an infinite loop, Apama forces the correlator to shut down.

**Note:** If a license file cannot be found, the correlator will run with reduced capabilities. See "Running Apama without a license file" in *Introduction to Apama*.

### Options

The `correlator` tool takes the following options:

Option	Description
<code>-V   --version</code>	Displays version information for the correlator.
<code>-h   --help</code>	Displays usage information.
<code>-p <i>port</i>   --port <i>port</i></code>	Specifies the port on which the event correlator should listen for monitoring and management requests. The default is 15903.
<code>-f <i>file</i>   --logfile <i>file</i></code>	Specifies the path of the status log file that the event correlator writes messages to. The default is <code>stdout</code> . See <a href="#">"Specifying log filenames" on page 139</a> and <a href="#">"Descriptions of correlator status log fields" on page 143</a> .
<code>-v <i>level</i>   --loglevel <i>level</i></code>	Specifies the level of information that the event correlator sends to the correlator status log file. In increasing amount of information order, the value must be one of the following: <code>OFF</code> , <code>CRIT</code> , <code>FATAL</code> , <code>ERROR</code> , <code>WARN</code> , <code>INFO</code> , <code>DEBUG</code> , <code>TRACE</code> . The default is <code>INFO</code> .
<code>-t   --truncate</code>	Specifies that if the correlator status log file already exists, the correlator should empty it first. The default is to append to it.
<code>-N <i>name</i>   --name <i>name</i></code>	Assigns a name to the correlator. The default is <code>correlator</code> . If you are running a lot of correlators you might find it useful to assign a name to each correlator. A name can make it easier to use the <code>engine_management</code> tool to manage correlators and adapters.
<code>-l <i>file</i>   --license <i>file</i></code>	Specifies the path to the license file.
<code>-m <i>num</i>   --maxoutstandingack <i>num</i></code>	Specifies that you want the correlator to buffer messages for up to <i>num</i> seconds for each receiver that the correlator sends events to. The default is 10. For additional information, see <a href="#">"Determining whether to disconnect slow receivers" on page 146</a> .

Option	Description
<code>-M num   --maxoutstandingkb num</code>	Specifies that you want the correlator to buffer the events for each receiver up to the size in kb represented by <i>num</i> .
<code>-x   --qdisconnect</code>	Specifies that you want the correlator to disconnect receivers that are consuming events too slowly. For details, see <a href="#">"Determining whether to disconnect slow receivers" on page 146</a> . The default is that the correlator does not disconnect slow receivers.
<code>--logQueueSizePeriod p</code>	Sets the interval at which the correlator sends information to its log file. The default is 5 seconds. Replace <i>p</i> with a <code>float</code> value for the period you want. <div> <b>Caution</b> Setting <code>logQueueSizePeriod</code> to 0 turns logging off. Without correlator logging information, it is impossible to effectively troubleshoot problems. See also <a href="#">"Descriptions of correlator status log fields" on page 143</a>. </div>
<code>--distMemStoreConfig dir</code>	Specifies that the distributed <code>MemoryStore</code> is enabled, using the configuration files in the specified directory. Note that the configuration filenames must end with <code>*-spring.xml</code> and the correlator will not start unless the specified directory contains at least one configuration file. For more information on a distributed <code>MemoryStore</code> 's configuration files, see <a href="#">"Using the Distributed MemoryStore" in <i>Developing Apama Applications in EPL</i></a> .
<code>--jmsConfig dir</code>	Specifies that correlator-integrated messaging is enabled using the configuration files in the specified directory. Note that the configuration filenames must end with <code>*-spring.xml</code> and the correlator will not start unless the specified directory contains at least one configuration file. For more information on the configuration files for correlator-integrated messaging for JMS, see

Option	Description
	"Configuration files" in <i>Connecting Apama Applications to External Components</i> .
<code>-j   --java</code>	Enables support for Java applications, which is needed to inject a Java application or plug-in using <code>engine_inject -j</code> .
<code>-J option   --javopt option</code>	<p>Specifies an option or property that you want the correlator to pass to the embedded JVM. You must specify the <code>-J</code> option for each JVM option. You can specify the <code>-J</code> option multiple times in the same <code>correlator.exe</code> command line. For example, <code>-J "-Djava.class.path=path" -J "-Da=value1" -J "-Db=value2" -J "-Xmx400m"</code></p> <p>When you use this option to pass the classpath to the JVM Apama prepends the correlator internal JAR files to the path you specify. If you set the <code>CLASSPATH</code> environment variable and also specify this option when you start the correlator, the path you specify in the correlator start-up command takes precedence.</p>
<code>--inputLog file</code>	<p>Specifies the path of the input log file. The event correlator writes only input messages to the input log file. If there is a problem with your application, Software AG Global Support can use the input log to try to diagnose the problem. An input log contains only the external inputs to the correlator. There is no information about multi-context ordering. Consequently, if there is more than one context, there is no guarantee that you can replay execution in the same order as the original execution. See <a href="#">"Replaying an input log to diagnose problems" on page 216</a>.</p>
<code>--XsetRandomSeed int</code>	<p>Starts the correlator with the random seed value you specify. Specify an integer whose value is in the range of 1 to <math>2^{32}</math>. The correlator uses the random seed to calculate the random numbers returned by the <code>integer.rand()</code> and <code>float.rand()</code> functions. The same</p>

Option	Description
	<p>random seed returns the same sequence of random numbers. This option is useful when your application uses the <code>integer.rand()</code> and <code>float.rand()</code> functions and you are using an input log to capture events and messages coming into a correlator. If you need to reproduce correlator behavior from that input log, you will want the correlator to generate the same random numbers as it generated when the original input was captured.</p>
<code>-XignoreEnqueue</code>	<p>For internal use only. Instructs the correlator to ignore <code>enqueue</code> statements when replaying an input log.</p>
<code>--inputQueueSize int</code>	<p>Sets the maximum number of spaces in every context's input queue. The default is that each input queue has 20,000 spaces. If events are arriving on an input queue faster than the correlator can process them the input queue can fill up. You can set the <code>inputQueueSize</code> option to allow all input queues to accept more events. Typically, the default input queue size is enough so if you find that you need to increase the size of the input queue you should try to understand why the correlator cannot process the events fast enough to leave room on the default-sized queue. If you notice that adapters or applications are blocking it might be because a public context's input queue is full. To determine if a public context's input queue is full, use output from the <code>engine_inspect</code> tool in conjunction with the status messages in the correlator log file.</p>
<code>-g   --nooptimize</code>	<p>Disables correlator optimizations that hinder debugging. Specify this option when you plan to run the <code>engine_debug</code> tool. You cannot run the <code>engine_debug</code> tool if you did not specify the <code>-g</code> option when you started the correlator.</p> <p>Software AG Designer automatically uses the <code>-g</code> option when it starts a correlator from a debug launch configuration. However, if</p>

Option	Description
	you are connecting to an externally-started correlator, and you want to debug in that correlator, you must ensure that the <code>-g</code> option was specified when the externally-started correlator was started.
<code>-P</code>	Enables correlator persistence. You must specify this option to enable correlator persistence. If you do not specify any other correlator persistence options, the correlator uses the default persistence behavior as described in "Enabling correlator persistence" in <i>Developing Apama Applications in EPL</i> . If you specify one or more additional correlator persistence options, the correlator uses the settings you specify for those options and uses the defaults for the other persistence options.
<code>-PsnapshotInterval=interval</code>	Specifies the period between persistence snapshots. The default is 200 milliseconds.
<code>-PadjustSnapshot=boolean</code>	Indicates whether or not the correlator should automatically adjust the snapshot interval according to application behavior. The default is true, which means that the correlator does automatically adjust the snapshot interval. You might want to set this to false to diagnose a problem or test a new feature.
<code>-PstoreLocation=path</code>	Specifies the path for the directory in which the correlator stores persistent state. The default is the current directory, which is the directory in which the correlator was started.
<code>-PstoreName=file</code>	Specifies the name of the file that contains the persistent state. This is the recovery datastore. The default is <code>persistence.db</code> .
<code>-Pclear</code>	Specifies that you want to clear the contents of the recovery datastore. This option applies to the recovery datastore you specify for the <code>-PstoreName</code> option or to the default <code>persistence.db</code> file if you do not specify the <code>-PstoreName</code> option. When the correlator

Option	Description
	starts it does not recover from the specified recovery datastore.
<code>-XrecoveryTime num</code>	For correlators that use an external clock, this is a time expression that represents the time of day that a correlator starts at when it recovers persistent state and restarts processing. The default is the time expression that represents the time of day captured in the last committed snapshot. This option is useful only for replaying input logs that contain recovery information. To change the default, specify a number that indicates seconds since the epoch.
<code>-noDatabaseInReplayLog</code>	Specifies that the correlator should not copy the recovery datastore to the input log when it restarts a persistence-enabled correlator. The default is that the correlator does copy the recovery datastore to the input log upon restarting a persistence-enabled correlator. You might set this option if you are using an input log as a record of what the correlator received. The recovery datastore is a large overhead that you probably do not need. Or, if you maintain an independent copy of the recovery datastore, you probably do not want a copy of it in the input log.
<code>--runtime mode</code>	<p>On Linux 64-bit systems, you can specify whether you want the correlator to use the compiled runtime or the interpreted runtime, which is the default. Specify <code>--runtime compiled</code> or <code>--runtime interpreted</code>.</p> <p>The interpreted runtime compiles EPL into bytecode whereas the compiled runtime compiles EPL into native code that is directly executed by the CPU. For many applications, the compiled runtime provides significantly faster performance than the interpreted runtime. Applications that perform dense numerical calculations are most likely to have the greatest performance improvement when the correlator uses the compiled runtime. Applications that spend most of their time</p>



Option	Description
	<p>managing listeners and searching for matches among listeners typically show a smaller performance improvement.</p> <p>Using the compiled runtime requires that the <code>binutils</code> package is installed on the Linux system.</p> <p>Other than performance, the behavior of the two runtimes is the same except for the following:</p> <ul style="list-style-type: none"> <li>■ The interpreted runtime allows for the profiler and debugger to be switched on during the execution of EPL. The compiled runtime does not permit this. For example, you cannot switch on the profiler or debugger in the middle of a loop.</li> <li>■ The amount of stack space available is different for the two runtimes. This means that recursive functions run out of stack space at a different level of recursion on the two runtimes.</li> </ul> <div data-bbox="721 1098 1339 1272"> <p><b>Note:</b> If you are using both correlator persistence (<code>-P</code> option) and the compiled runtime (<code>--runtime compiled</code> option), we recommend the use of the <code>--runtime-cache</code> option to improve recovery times.</p> </div>
<code>--runtime-cache <i>dir</i></code>	<p>Enables caching of compiled runtime objects in the specified directory. Subsequent injections of the same files to any correlator using that cache will be quicker. For more information, see <a href="#">"Injection time of compiled runtime" on page 149</a>.</p>
<code>--frequency <i>num</i></code>	<p>Instructs the correlator to generate clock ticks at a frequency of <i>num</i> per second. Defaults to 10, which means there is a clock tick every 0.1 seconds. Be aware that there is no value in increasing <i>num</i> above the rate at which your operating system can generate its own clock ticks internally. On UNIX and some Windows machines this is 100 and on other Windows machines it is 64.</p>

Option	Description
<code>-Xclock   --externalClock</code>	Instructs the correlator to disable its internal clock. By default, the correlator uses internally generated clock ticks to assign a timestamp to each incoming event. When you specify the <code>-Xclock</code> option, you must send time events ( <code>&amp;TIME</code> ) to the correlator. These time events set the correlator's clock. For additional information, see <a href="#">"Determining whether to disable the correlator's internal clock" on page 148</a> .
<code>-Xconfig file   --configFile file</code>	Specifies a special configuration file that the correlator uses to initialize its networking. Specify this option only as directed by Apama Technical Support to troubleshoot networking problems. For more information, see <a href="#">"Using the Apama component extended configuration file" on page 225</a> .
<code>-r list   --rnames list</code>	<p>Specifies one or more Universal Messaging (UM) realm names that you want this correlator to connect to. Separate names with commas. See "Starting correlators that use UM" in <i>Connecting Apama Applications to External Components</i>.</p> <p>If you specify the <code>--rnames</code> option and also the <code>--umConfigFile</code> option, the specified UM configuration file takes precedence.</p>
<code>-UMconfig file   --umConfigFile file</code>	<p>Specifies the name of a properties file that defines the UM configuration for this correlator. See "Defining UM properties for Apama applications" in <i>Connecting Apama Applications to External Components</i>.</p> <p>If you specify the <code>--rnames</code> option and also the <code>--umConfigFile</code> option, the specified UM configuration file takes precedence.</p>
<code>--connectivityConfig file</code>	Specifies the path to the configuration file that specifies connectivity plug-ins to send and receive messages. For more information, see "Using Connectivity Plug-ins" in <i>Connecting Apama Applications to External Components</i> .

Option	Description
<code>--config <i>file</i></code>	Available as of 9.10.0.4. Specifies the name of a .yaml file that is used to configure the correlator. This option may only be specified once. See also <a href="#">"About deploying Apama applications with a YAML configuration file" on page 23</a> .

### Exit status

The `correlator` tool returns the following exit values:

Value	Description
0	The event correlator terminated successfully.
1	An error occurred which caused the event correlator to terminate abnormally.

## Specifying log filenames

A correlator can send information to the following log files:

- **Correlator status log file.** Upon correlator startup, the default behavior is that the correlator logs status information to `stdout`. To send correlator status information to a file, specify the `-f file` or `--logfile file` option and replace *file* with a log filename. The format for specifying a log filename is described below.  
  
Before you specify a log filename, you should consider your log rotation policy, which can determine what you want to specify for the log filename. See ["Rotating the correlator log file" on page 197](#).
- **Correlator input log file.** When you start a correlator, you can specify the `--inputlog file` option so that the correlator maintains a special log file for all inputs. Again, before you specify the log filename, consider the rotation policy for your input log files. See ["Rotating an input log file" on page 216](#).
- **Application log files.** You can create log files for packages, monitors, and events in your application. The format you use to specify these log filenames is the same as for status logs and input logs. For details about how to create application log files, see ["Setting logging attributes for packages, monitors and events" on page 194](#).

The format for specifying a log filename is as follows:

```
file [${START_TIME}][${ROTATION_TIME}][${ID}][${PID}].log
```

The following table describes each part of a log filename specification. You cannot include spaces. You can separate the parts of the filename specification with

underscores. You can specify `${START_TIME}`, `${ROTATION_TIME}`, `${ID}` and/or `${PID}` in any order. For examples, see ["Examples for specifying log filenames" on page 141](#).

Element	Description
<i>file</i>	<p>Replace <i>file</i> with the name of the file that you want to be the log file. If you specify the name of a file that exists, the correlator overwrites it on Windows or appends to it on UNIX.</p> <p>Required.</p> <p>If you also specify <code>\${START_TIME}</code> and/or <code>\${ROTATION_TIME}</code> and/or <code>\${ID}</code> and/or <code>\${PID}</code>, the correlator prefixes the name you specify to the time the correlator was started and/or the time the log file was rotated (logging to a new file began) and/or an ID beginning with 001 and/or the process ID.</p> <p>Be sure to specify a location that allows fast access.</p>
<code>\${START_TIME}</code>	<p>Tag that indicates that you want the correlator to insert the date and time that it started into the log filename.</p> <p>Optional, however you probably want to always specify either this option or <code>\${ROTATION_TIME}</code> to avoid overwriting log files.</p> <p>This tag is also useful for correlators that you start from Apama's Management and Monitoring console, because it lets you distinguish the logs from different correlators.</p>
<code>\${ROTATION_TIME}</code>	<p>Tag that indicates that you want the correlator to insert the date and time that it starts sending messages to a new log file into the log filename.</p> <p>Optional.</p> <p>If you specify <code>\${ROTATION_TIME}</code> and this log filename specification appears in a correlator start-up command then the name of the initial log file contains the time the correlator started.</p> <p>This tag is also useful for correlators that you start from Apama's Management and Monitoring console, because it lets you distinguish the status logs from different correlators.</p>

Element	Description
<code>\${ID}</code>	<p>Tag that indicates that you want the correlator to insert a three-digit ID into the log filename. The ID that the correlator inserts first is 001.</p> <p>Optional. The log ID increment is related only to rotation of log files. See <a href="#">"Rotating the correlator log file" on page 197</a> and <a href="#">"Rotating an input log file" on page 216</a>.</p> <p>The ID allows you to create a sequence of log files. Each time the log file is rotated, the correlator increments the ID. A sequence of log files have the same name except for the ID. If you also specify <code>\${ROTATION_TIME}</code> then a sequence of log files have the same name except for the rotation time and the ID.</p> <p>Restarting the correlator always resets the ID portion of the log filename to 001.</p>
<code>\${PID}</code>	<p>Tag that indicates that you want the correlator to insert the process ID into the log file name.</p> <p>Optional.</p> <p>The process ID will be constant for the lifetime of the process. Therefore, if you start multiple processes with the same arguments, they get different file names.</p>

If you plan to rotate log files then be sure to specify `${ROTATION_TIME}` or `${ID}`. You can also specify both.

## Examples for specifying log filenames

This topic provides examples of specifying log filenames. The format for specifying a log filename is the same in the following cases:

- Starting the correlator and specifying a correlator status log file with the `--logfile` option.
- Starting the correlator and specifying a correlator input log file with the `--inputLog` option.
- Invoking `engine_management --setLogFile` to rotate the correlator status log.
- Invoking `engine_management --setApplicationLogFile` to create a log file for a package, monitor or event.

The following specifies that the name of the status log is `correlator_status.log`:

```
--logfile correlator_status.log
```

Suppose that the correlator processes events for a while, sends information to `correlator_status.log`, and then you find that you need to restart the correlator. If you restart the correlator and specify the exact same log filename, the correlator overwrites the first `correlator_status.log` file. To avoid overwriting a status log, specify `${START_TIME}` in the log file name specification when you start the correlator. For example:

```
--logfile correlator_status_${START_TIME}.log
```

The above command opens a status log with a name something like the following:

```
correlator_status_2015-03-12_15:12:23.log
```

This ensures that the correlator does not overwrite a log file. Now suppose that you want to be able to rotate the log, so you specify the `${START_TIME}` and `${ID}` tags:

```
correlator_status_${START_TIME}_${ID}.log
```

The initial name of the log file is something like the one on the first line below. If you then rotate the log file then the correlator closes that file and opens a new file with a name like the one on the second line:

```
correlator_status_2015-03-12_15:12:23_001.log
correlator_status_2015-03-12_15:12:23_002.log
```

To specify an application log filename for messages generated in `com.example.mypackage`, you can specify the log filename as follows:

```
mypackage_${ID}_${ROTATION_TIME}.log
```

With that specification, messages generated in `com.example.mypackage` will go to a file with a name such as the one on the first line below. The time in the initial application log filename is the time that the initial log file is created. If you rotate the logs every 24 hours at midnight then the names of subsequent application log files will be something like the names in the second and third lines below.

```
mypackage_001_2015-03-21_18:42:06.log
mypackage_002_2015-03-22_00:00:00.log
mypackage_003_2015-03-23_00:00:00.log
```

If you want to run multiple correlators with the same arguments but with separate log files, you can use the process ID to differentiate them:

```
--logfile correlator_${PID}.log
```

The above command will produce a log file with a name such as the following, where each correlator will have a unique log file:

```
correlator_23487.log
```

### UNIX note

In most UNIX shells, when you start a correlator you most likely need to escape the tag symbols, like this:

```
correlator -l license --inputLog input_\${START_TIME}_\${ID}.log
```

## Descriptions of correlator status log fields

The correlator sends information to its status log file every five seconds (the default behavior) or at an interval that you specify with the `--logQueueSizePeriod` option when you start the correlator. When logging at `INFO` level, this information contains the following:

```
Status: sm=2 nctx=2 ls=3 rq=0 eq=0 iq=0 oq=0 icq=12 lcn="input ctx one"
lcq=12 lct=0.8 rx=5 tx=20 rt=7 nc=1 vm=369768 pm=956240 runq=0 si=0.0
so=0.0 srn="apamacluster1_node3" srq=3
```

Where the fields have the following meanings:

Field	Description
sm	Sub-Monitors. Number of monitor instances, sometimes called sub-monitors. This is the sum of monitor instances in the main context plus monitor instances in any other context.
nctx	Number of contexts. For applications developed prior to Apama 4.1, this is always 1.
ls	Listener sum. This is the number of listeners in the main context plus the number of listeners in any other context. This includes each <code>on</code> statement and each stream source template, for example, <code>all Tick(symbol="APMA")</code> in the following: <pre>stream&lt;Tick&gt; ticks := all Tick(symbol="APMA");</pre>
rq	Route queue. Sum of the number of routed events on the input queues of all contexts. A routed event is any event that has been generated by EPL's <code>route</code> keyword or JMon's <code>route()</code> method. A routed event goes to the front of that context's input queue. This ensures that the correlator processes routed events before processing external events. This number can go up and down, and it tends to be 0 for an idle correlator.
eq	Enqueue queue. Number of events on the enqueued events queue. An enqueued event is any event that has been generated with the EPL <code>enqueue</code> keyword (not <code>enqueue...to</code> ) or JMon <code>enqueue()</code> method. A separate thread moves events from the enqueued events queue to the input queue of each public context. The size of the enqueued events queue is unbounded. Consequently, it is possible for this queue to use a large amount of memory if one or more input queues are full.
iq	Input queue. Sum of the number of entries on the input queues of all contexts. This number excludes routed events. It includes events

Field	Description
	from external sources, injections of EPL files, delete requests, time ticks, pending spawn-to operations, and enqueued events. This number goes up and down, and tends to be 0 for an idle correlator. It is possible for the total number of input queue entries ( <code>iq</code> ) to be greater than the number of events the correlator has received from external sources ( <code>rx</code> ).
<code>oq</code>	Output queue. Number of events on the correlator's output queue. This is the number of events that the correlator has emitted but not yet sent to any receivers. If the correlator is idle, this number is 0.
<code>icq</code>	Sum of enqueued events on the input queues of all public contexts. These events are a subset of the events included in the <code>iq</code> count.
<code>lcn</code>	Name of the slowest context. This is the context most backed up in time. If no context has entries on its input queue, then this value is "<none>". The name of the main context is always <code>main</code> .
<code>lcq</code>	For the context identified by <code>lcn</code> , this is the number of entries on its queue.
<code>lct</code>	For the context identified by <code>lcn</code> , this is the time difference between its current logical time and the most recent time tick added to its input queue.
<code>rx</code>	<p>Number of events the correlator has received from external sources since the correlator was started. The correlator increments this count as soon as it receives an event. After incrementing this count, the correlator parses the event to determine if it is an event for which a definition has already been injected. If the correlator can successfully parse the event, the event goes to the input queue of each public context. If the correlator cannot parse the event, the correlator discards the event.</p> <p>This is not the number of events that the correlator has processed. This count does not include routed and enqueued events.</p> <p>This number never goes down; it can only go up.</p>
<code>tx</code>	Number of events the correlator has sent to receivers since the correlator was started. This number includes duplicate events sent to multiple receivers. For example, suppose you inject EPL code that emits an event, and there are five receivers that are subscribed to channels that publish that event. In this situation, the <code>tx</code> count goes



Field	Description
	up by five. Although there was 1 event, it was sent five times: once to each subscribed receiver.
<code>rt</code>	Route total. Total number of events that have been routed across all contexts since the correlator was started.
<code>nc</code>	Number of receivers.
<code>vm</code>	Virtual memory. Number of kilobytes of virtual memory being used by the correlator process.
<code>pm</code>	Physical/resident memory. Number of kilobytes of physical/resident memory being used by the correlator process.
<code>runq</code>	Run queue. Number of contexts (public and private) that have work to do but are not currently running. These contexts are waiting for processing resources. This indicator is a measure of the load on the system. When this number is consistently more than 0 and latency is a problem you might want to consider adding CPUs to your configuration.
<code>si</code>	The rate (pages per second) at which pages are being read from swap space.
<code>so</code>	The rate (pages per second) at which pages are being written to swap space.
<code>srn</code>	Slowest receiver name. Name of the receiver whose queue has the largest number of entries. If no receivers have queue entries, then this value is "<none>".
<code>srq</code>	Slowest receiver queue. For the receiver identified by <code>srn</code> , the slowest receiver, this is the number of entries on its queue.

The above table uses the term "receiver". See ["Watching correlator runtime status" on page 164](#) for information on the possible types of receivers.

Correlators with correlator-integrated messaging for JMS enabled send additional information to the status log file. For details on this information, see "Logging correlator-integrated messaging for JMS status" in *Connecting Apama Applications to External Components*.

## Text internationalization issues

Apama translates the contents of the correlator status log from UTF-8 to the local character set before displaying it in a console or terminal.

## Determining whether to disconnect slow receivers

The correlator sends events to multiple receivers. Sometimes, a receiver cannot consume its events fast enough for the correlator to continue sending them. When this happens, the default behavior is that the correlator suspends processing until the receiver disconnects or starts consuming events fast enough. In other words, a slow receiver can prevent other consumers from receiving events. However, you might prefer to have the correlator disconnect a slow receiver and continue processing and sending events to other consumers. Information in this section can help you determine whether to disconnect slow receivers.

See also "Handling slow or blocked receivers", in the testing and tuning section of *Developing Apama Applications*.

## Description of slow receivers

Every event that the correlator sends to one of its receivers has a sequence number. After a receiver processes an event, it sends the event's sequence number back to the correlator as an acknowledgment that the receiver processed that event. By default, if the correlator does not receive an acknowledgment within 10 seconds after the correlator sent the event, the correlator marks that receiver as being slow to consume events.

You can control the length of time within which the receiver must acknowledge an event before it is marked as a slow receiver. When you start the correlator, you can specify the `-m` (or `--maxoutstandingack`) option and specify a number.

For example:

```
correlator -l ApamaServerLicense.xml -m 15
```

If you start the correlator with this command, the correlator marks a receiver as slow if the correlator does not receive an acknowledgment within 15 seconds. If you do not specify the `-m` option, the default is 10 seconds. You should not specify a value under 1 second because doing so raises the risk that the correlator might designate a receiver as slow when it is in fact not slow.

The mechanism that flags a receiver as slow is not precise. If a receiver does not acknowledge an event sequence after 10 seconds (the default setting), the correlator does not immediately designate the receiver as slow. Typically, the designation happens within the next 5 seconds. If you change the value of the `-m` option, the slow designation takes effect between 1 and 1.5 times the value of the `-m` option.

## How frequently slow receivers occur

In practice, sending acknowledgments should not be slow because a dedicated thread sends acknowledgments. Network interruptions are the most common cause of delayed acknowledgments. Of course, network interruptions affect events being sent as well.

Most receivers, including the `engine_receive` tool, normally send acknowledgments 0.1 seconds after the message was sent. Consequently, there is very little chance of a receiver being mistakenly designated as slow. In production, slow receivers should be rare as long as you have done the appropriate load testing before deployment.

If an engine library client blocks in the middle of a `sendEvents` call, the receiver cannot acknowledge messages while the client is blocked. As you know, a receiver is made up of an engine library and a client. Clients receive events by registering a `sendEvents` callback with the engine library. When the engine library gets an event from the correlator, it calls `sendEvents`. Problems that can cause a client to block are typically related to I/O, networking, or synchronization. The `sendEvents` call cannot complete until the problem is resolved. The receiver cannot send the acknowledgment until the `sendEvents` call completes. For example, the `engine_receive` tool is a receiver that is made up of an engine library and a client whose `sendEvents` callback writes events received to a disk file. If the client has to wait for the disk, then it is blocked. The likelihood of a `sendEvents` callback being blocked depends on what the client is doing. If the client is writing to a local disk, the process might block sometimes, but never more than a fraction of a second. However, sending the events over a slow or unreliable network might block for a while if the network, or the remote system cannot keep up with the event rate.

During development of event consumers, however, slow receivers are more likely. This can happen when a newly developed consumer receives an event from the correlator but cannot send the acknowledgment because of a deadlock. Another development problem might be that the event consumer cannot keep up with the load. If you have problems with slow receivers during development, you probably need to evaluate the design of your application.

## Correlator behavior when there is a slow receiver

When the correlator has a slow receiver, it can behave in one of two ways:

- The default behavior is that the correlator blocks further processing. This is because a slow receiver causes the correlator's event output queue to become full. When the queue is full, the correlator stops processing because it has no place to put events. The processing thread stops and does not execute any more EPL code. The transport thread does not send any more events to any of the correlator's other receivers. The correlator resumes processing when the slow receiver disconnects or acknowledges the outstanding sequence number.
- The correlator disconnects the slow receiver, and continues processing events and sending them to its other receivers. To obtain this behavior, you specify the `-x` (or `--qdisconnect`) option when you start the correlator. The correlator sends a message

to the receiver to indicate that the correlator is disconnecting the receiver. It is up to the receiver to reconnect.

To ensure that it has received an acknowledgment for every event sent, the correlator buffers each event that it sends until it receives the message's corresponding acknowledgment. When there is a slow receiver, this can use a lot of memory if the correlator is sending a large number of messages.

## Tradeoffs for disconnecting a slow receiver

When you specify the `-x` option when you start the correlator, it means that the correlator always disconnects a slow receiver. There are two main disadvantages to this:

- The correlator loses the events that it sent to that receiver.
- It is possible for the correlator to disconnect a receiver that is temporarily overloaded, and to therefore lose events unnecessarily.

Clearly, losing events can be a very serious problem. This is why the default is that the correlator does not disconnect slow receivers.

The advantage of disconnecting a slow receiver is that the correlator continues processing events.

The correlator always sends a warning message to its status log when it detects a slow receiver. This lets you see where there are potential problems.

If you cannot allow the correlator to lose events, do not specify the `-x` option when you start the correlator.

## Determining whether to disable the correlator's internal clock

When you start the correlator, you can specify the `-Xclock` option to disable the correlator's internal clock. By default, the correlator uses internally generated clock ticks to assign a timestamp to each incoming event. When you specify the `-Xclock` option, you must send time events (`&TIME`) to the correlator. These time events set the correlator's clock.

Use `&TIME` events in place of the correlator's internal clock when you want to reproduce the historic behavior of an application. For example, Apama's Data Player in Software AG Designer starts a correlator with a command that specifies the `-Xclock` option. The Data Player then sends `&TIME` events that let you play back events from the database.

A situation in which you might want to generate `&TIME` events is when you want to run experiments at faster than real time but still obtain correct timestamp behavior. In this situation, the correlator uses the externally generated time events instead of real time to invoke timers and wait events.

Disabling the correlator's internal clock, and sending external time events, affects all temporal operations, such as timers and `wait` statements.

Regardless of whether the correlator generates internal clock ticks, or receives external time events, the correlator assigns a timestamp to each incoming event. The timestamp indicates the time that the event arrived on the context's input queue. The value of the

timestamp is the same as the last internally-generated clock tick or externally-generated time event. For example, suppose you have the following events and clock ticks:

```
&TIME (1)
A ()
B ()
&TIME (2)
C ()
```

A and B receive a timestamp of 1. C receives a timestamp of 2.

For additional information about using external time events, see "Apama Concepts" in *Introduction to Apama*.

## Injection time of compiled runtime

Injection times for systems using the compiled runtime can be very long - significantly longer than if using only the interpreted runtime. Subsequent injection times can be improved by using the `--runtime-cache dir` option, which specifies a directory where the correlator can cache the compilation state (see ["Starting the event correlator" on page 130](#)). This stores the results of compiling EPL code on disk to be used for subsequent injections of the same code.

The compiled EPL code is specific to the system on which it was compiled and the version of Apama that was used to compile it. This means that while a cache can be moved or shared between machines to improve startup on a new machine, it must be identical to the original. Otherwise, the cached version cannot be used and it must be recompiled.

An injection is able to use a cached version of a previous injection if all of the following are the same as in a previous injection:

- The EPL source code.
- The source code of all files that contain any type that an injection depends on.
- The correlator version.
- The host operating system.
- The CPU model.

The results of injections can be affected by any of the above. Therefore, if any change occurs, the correlator will re-compile the EPL.

The cache is designed to be used to speed up re-injection on production systems and not for quicker development cycles, which should typically use the interpreter for faster injection times. If there are identical user acceptance testing (UAT) and production environments, then the UAT environment can prime a cache which can then be used by the production correlator to improve initial startup times. However, the two systems must be identical. The strings used to disambiguate systems are logged at correlator startup when using the compiled runtime and can be used to compare the systems.

The cache contents are never removed by the correlator. If you change your source, correlator versions or platform, then the cache may grow and contain stale items which

are no longer needed. If cache sizes become a problem, then we either recommend deleting the entire cache, or just the cache files with the oldest timestamps. The correlator will transparently recompile any needed files which are missing from the cache.

## Injecting code into a correlator

To inject EPL files, JMon applications, or correlator deployment packages (CDPs) into the correlator, invoke the `engine_inject` tool. The executable for this tool is located in the `bin` directory of the Apama installation. Running the tool in the Apama Command Prompt (see ["Setting up the environment using the Apama Command Prompt" on page 26](#)) ensures that the environment variables are set correctly.

### Synopsis

To inject applications into the event correlator, run the following command:

```
engine_inject [ options ] [ file1 [ file2 ... ] ]
```

When you run this command with the `-h` option, the usage message for this command is shown.

### Description

The `engine_inject` tool reads application definitions from the specified file(s) and injects them into an event correlator. If you do not specify a filename, or if you specify a hyphen (-) as the filename, the correlator reads data from the standard input device (`stdin`) until you indicate the end of the file: CTRL+D on UNIX and CTRL+Z on Windows.

Application definitions can be monitors scripted in Apama's Event Processing Language. For more information on EPL, see "Introduction to Apama Event Processing Language" in *Developing Apama Applications*. Alternatively, you can specify the `-j` or `-c` options. The `-j` option specifies that you will inject an application written in Java. The `-c` option specifies that you will inject a correlator deployment package file.

When you specify the `-j` option, each file you inject must be a Java archive file (JAR) that contains a single JMon application. For more information, see "Overview of JMon Applications" in *Developing Apama Applications*.

When you specify the `-c` option, the file you inject must be an Apama correlator deployment package (CDP). For more information on preparing a CDP, see ["Packaging EPL and Java code" on page 156](#).

By default, the `engine_inject` tool is silent unless an error occurs. To view information about `engine_inject` execution, specify the `--verbose` option.

If you try to inject invalid EPL files or invalid JMon applications, the event correlator generates an error. None of the application data in the invalid file is loaded. The `engine_inject` tool terminates. If you specify multiple EPL or Java files for injection the

`engine_inject` tool injects all of them or terminates when it reaches the first file that contains an error. For example:

```
engine_inject 1.mon 2.mon 3.mon
```

If the `2.mon` file contains an error, then `engine_inject` successfully injects `1.mon` and then terminates when it finds the error in `2.mon`. The tool does not operate on `3.mon`.

If you try to inject a CDP, the correlator processes each EPL file packaged in the CDP separately. If one file in a CDP contains an error, then the correlator reports an error for that file and does not run it but it does run the other files in the CDP (if they have no errors). It does not matter which file in the CDP contains the error. That is, the first file in the CDP that the correlator processes can contain an error and the correlator still runs the other files in the CDP if they contain no errors.

**Note:** If a license file cannot be found, the correlator does not allow the injection of user-generated CDPs. See "Running Apama without a license file" in *Introduction to Apama*.

## Options

The `engine_inject` tool takes the following options:

Option	Description
<code>-h   --help</code>	Displays usage information.
<code>-n host   --hostname host</code>	Name of the host on which the event correlator is running. The default is <code>localhost</code> . Non-ASCII characters are not allowed in host names.
<code>-p port   --port port</code>	Port on which the event correlator is listening. The default is <code>15903</code> .
<code>-v   --verbose</code>	Requests verbose output during <code>engine_inject</code> execution.
<code>-u   --utf8</code>	Indicates that input files are in UTF-8 encoding. The default is that the <code>engine_inject</code> tool assumes that the EPL files to be injected are in the native character set of your platform. Set this option to override this assumption. The <code>engine_inject</code> tool then assumes that all input files are in UTF-8.
<code>-V   --version</code>	Displays version information for the <code>engine_inject</code> tool.

Option	Description
<code>-j</code>   <code>--java</code>	Indicates that each operand is a Java archive file (JAR file) that contains a single JMon application.
<code>-c</code>   <code>--cdp</code>	Indicates that each operand is a correlator deployment package (CDP) file.
<code>-s</code>   <code>--hashes</code>	Indicates that instead of injecting the specified files you want to print the hashes (UTF8-encoded) for the files. If <code>engine_inject</code> is operating on Java or correlator deployment package (CDP) files, then you must also specify <code>-j</code> or <code>-c</code> .

### Operands

The `engine_inject` tool takes the following operands:

Operand	Description
<code>[ file1 [ file2 ... ] ]</code>	The names of zero or more files that contain application data in Apama EPL, JMon, or correlator deployment package (CDP) files. If you do not specify one or more filenames, the <code>engine_inject</code> tool takes input from <code>stdin</code> .

### Exit status

The `engine_inject` tool returns the following exit values:

Value	Description
0	All definitions were injected into the event correlator successfully.
1	No connection to the event correlator was possible or the connection failed.
2	Other error(s) occurred while injecting the supplied definitions.

### Text encoding

By default, the `engine_inject` tool uses the default system encoding to determine the local character set. The `engine_inject` tool then translates all submitted EPL text



from the local character set to UTF-8. Consequently, it is important to correctly set the machine's locale.

However, some input files might start with a UTF-8 Byte Order Mark. The `engine_inject` tool treats such input files as UTF-8 and does not do any translation. Alternatively, you can specify the `-u` option when you run the `engine_inject` tool. This forces the tool to treat each input file as UTF-8.

## Deleting code from a correlator

The `engine_delete` tool removes EPL code and JMon applications from the correlator. The executable for this tool is located in the `bin` directory of the Apama installation. Running the tool in the Apama Command Prompt (see ["Setting up the environment using the Apama Command Prompt" on page 26](#)) ensures that the environment variables are set correctly.

### Synopsis

To remove applications from the event correlator, run the following command:

```
engine_delete [ options ] [ name1 [ name2 ... ] ]
```

When you run this command with the `-h` option, the usage message for this command is shown.

### Description

The `engine_delete` tool deletes named applications, monitors and event types from an event correlator. Names are the full package names as previously assigned to an application monitor or event type when injected into the event correlator.

To specify the items you want to delete, you can specify any one of the following in the `engine_delete` command line:

- Names of the items to delete.
- The `-f` option with the name of a file that contains the names of the items you want to delete. In this file, specify each name on a separate line.
- Neither of the above. In this case, the `engine_delete` tool reads names from `stdin` until you type an end-of-file signal, (CTRL+D on UNIX and CTRL+Z on Windows). If you want, you can specify a hyphen (-) in the command line to indicate that input will come from `stdin`.

The tool is silent by default unless an error occurs. To receive progress information, specify the `-v` option.

The tool permits two kinds of operations: delete and kill. These cause different side-effects. Therefore, you must use them carefully.

- When you delete a monitor, the correlator tries to terminate all of that monitor's instances. If they are responsive (not in some deadlocked state), each one executes its `ondie()` action, and when the last one exits the correlator calls the monitor's

`onunload()` action. This assumes that the monitor you are deleting defines `ondie()` and `onunload()` actions.

If a monitor instance does not respond to a delete request, the correlator cannot invoke the monitor's `onunload()` action. In this case, you must kill, rather than delete, the monitor instance.

- When you kill a monitor, the correlator immediately terminates all of the monitor's instances, without invoking `ondie()` or `onunload()` actions.

## Options

The `engine_delete` tool takes the following command line options:

Option	Description
<code>-h   --help</code>	Displays usage information. Optional.
<code>-n host   --hostname host</code>	Name of the host on which the event correlator is running. The default is <code>localhost</code> . Optional. Non-ASCII characters are not allowed in host names.
<code>-p port   --port port</code>	Port on which the event correlator is listening. Optional. The default is <code>15903</code> .
<code>-f filename   --file filename</code>	Indicates that you want the <code>engine_delete</code> tool to read names of items to delete from the specified file. In this file, each line contains one name. Optional. The default is that input comes from <code>stdin</code> .
<code>-F   --force</code>	Forces deletion of named event types even if they are still in use. That is, they are referenced by active monitors or applications. A forced delete also removes all objects that refer to the event type you are deleting. For example, if monitor <code>A</code> has listeners for <code>B</code> events and <code>C</code> events and you forcibly delete <code>C</code> events, the operation deletes monitor <code>A</code> , which of course means that the listener for <code>B</code> events is deleted. Optional. The default is that event types that are in use are not deleted.
<code>-k   --kill</code>	Kills all instances of the named monitor regardless of whether an instance is in use. For example, you can specify this option to remove a monitor that is stuck in an infinite loop. Any

Option	Description
	<code>ondie()</code> and <code>onunload()</code> actions defined in killed monitors are not executed.
<code>-a   --all</code>	Forces deletion of all applications, monitors, and event types. The correlator finishes processing any events on input queues and then does the deletions. Any events sent after invoking <code>engine_delete -a</code> are not recognized. Specifying this option does not stop a monitor that is in an infinite loop. You must explicitly kill such monitors. Specifying the <code>-a</code> option is equivalent to specifying the <code>-F</code> option and naming every object in the correlator. If you want to kill every object in the correlator, shut down and restart the correlator. See <a href="#">"Shutting down and managing components" on page 172</a> .
<code>-y   --yes</code>	Removes the "are you sure?" prompt when using the <code>-a</code> option.
<code>-v   --verbose</code>	Requests verbose output.
<code>-u   --utf8</code>	Indicates that input files are in UTF-8 encoding. This specifies that the <code>engine_delete</code> tool should not convert the input to any other encoding.
<code>-V   --version</code>	Displays version information for the <code>engine_delete</code> tool.

## Operands

The `engine_delete` tool takes the following operands:

Operand	Description
<code>[ name1 [ name2 ... ] ]</code>	The names of zero or more EPL or JMon applications, monitors and/or event types to delete from the event correlator. If you do not specify at least one item name, and you do not specify the <code>-f</code> option, the <code>engine_delete</code> tool expects input from <code>stdin</code> .

**Exit status**

The `engine_delete` tool returns the following exit values:

Value	Description
0	The items were deleted from the event correlator successfully.
1	No connection to the event correlator was possible or the connection failed.
2	Other error(s) occurred while deleting the named items.

**Packaging EPL and Java code**

The `engine_package` tool assembles EPL files and JAR files into a correlator deployment package (CDP). You can inject a CDP file into the correlator just as you inject an EPL file or a JAR file containing a JMon application. CDP files use a proprietary, non-plaintext format that treats files in a manner similar to the way a JAR file treats a collection of Java files. In addition, using a CDP file guarantees that all files, assuming no errors, are injected and are injected in the correct order. See ["Injecting code into a correlator" on page 150](#) for details about how the correlator handles an error in a file that is in a CDP.

While the names of events, monitors, aggregates, and JAR files that are contained in a CDP file are visible to the correlator utilities `engine_inspect`, `engine_manage`, and `engine_delete`, the code that defines them is not.

The executable for this tool is located in the `bin` directory of the Apama installation. Running the tool in the Apama Command Prompt (see ["Setting up the environment using the Apama Command Prompt" on page 26](#)) ensures that the environment variables are set correctly.

**Synopsis**

To package files into a CDP file, run the following command:

```
engine_package [ options ] [ file1 [ file2 ... ] ]
```

When you run this command with the `-h` option, the usage message for this command is shown.

**Description**

The `engine_package` tool creates a correlator deployment package (CDP). A CDP file contains one or more files. You specify the name of the CDP file to create as an argument to the `-o` option.

You can specify the files you want to include on the command line, or you can use the `-m` option and specify a manifest file that contains the names of the files. The manifest file is a text file; each line in the file specifies a relative or absolute path to a file. Files should be listed in the order in which you want them to be injected into the correlator.

### Options

The `engine_package` tool takes the following options:

Option	Description
<code>-h   --help</code>	Displays usage information.
<code>-V   --version</code>	Displays version information for the <code>engine_package</code> tool.
<code>-o filename   --output filename</code>	Name of the CDP file to create. Required.
<code>-m filename   --manifest filename</code>	Name of the manifest file that lists the files you want to package.
<code>-u   --utf8</code>	Indicates that input files are in UTF-8 encoding. The default is that the <code>engine_package</code> tool assumes that the files to be packaged are in the native character set of your platform. Set the <code>-u</code> option to override this assumption. The <code>engine_package</code> tool then assumes that all input files are in UTF-8.

### Operands

The `engine_package` tool takes the following operands:

Operand	Description
<code>[ file1 [ file2 ... ] ]</code>	The names of the EPL or JAR files that contain code. The order in which these files are specified will become the order in which they are injected into the correlator when the CDP file is injected. Instead of listing the files on the command line, you can list them in a manifest file and use the <code>-m</code> option.

**Exit status**

The `engine_package` tool returns the following exit values:

Value	Description
0	Returned on success.
1	Returned on any error.

**Example**

The following example describes how to create a correlator deployment package file with multiple monitor files and inject the CDP file into a running correlator.

1. Create a manifest file containing a list of files to include in the CDP. For this example, the file is named `manifest.txt` and each line contains the full path name of an EPL file or JAR file:

```
c:\dev\sample\monitor1.mon
c:\dev\sample\monitor2.mon
C:\dev\sample\jmon-app.jar
```

2. To create the CDP file, call the `engine_package` tool stating the output filename and the manifest file to include in the CDP. (Note, instead of using a manifest file, you can list the files individually in the `engine_package` arguments.)

```
engine_package.exe -o c:\sample.cdp -m c:\dev\sample\manifest.txt
```

3. To inject the CDP file, call the `engine_inject` tool with `-c` (or `--cdp`). This injects each file that is included in the CDP file into the correlator.

```
engine_inject.exe -c c:\sample.cdp
```

Sample output from the correlator:

```
2012-07-11 13:51:33.156 INFO [3852] - Injected CDP from file
c:\sample.cdp (b2f097b02791e5dd4ac73cda38e153e9),
size 313 bytes, decoding and compile time 0.00 seconds
```

## Sending events to correlators

The `engine_send` tool sends Apama-format events into an event correlator or IAF adapter. The executable for this tool is located in the `bin` directory of the Apama installation. Running the tool in the Apama Command Prompt (see ["Setting up the environment using the Apama Command Prompt" on page 26](#)) ensures that the environment variables are set correctly.

If the events you want to send are not in Apama format, you must use an adapter that can transform your event format into Apama event format.

## Synopsis

To send Apama-format events to an event correlator or IAF adapter, run the following command:

```
engine_send [ options ] [ file1 [ file2 ... ] ]
```

When you run this command with the `-h` option, the usage message for this command is shown.

## Description

The `engine_send` tool sends Apama-format events to an event correlator. In Apama-format event files, you can specify whether to send the events in batches of one or more events or at set time intervals.

The correlator reads events from one or more specified files. Alternatively, you can specify a hyphen (-) or not specify a filename so that the correlator reads events from `stdin` until it receives an end-of-file signal (CTRL+D on UNIX and CTRL+Z on Windows).

For details about Apama-format events, see ["Event file format" on page 219](#).

By default, the `engine_send` tool is silent unless an error occurs. To view progress information during `engine_send` execution, specify the `-v` option when you invoke `engine_send`.

You can also use `engine_send` to send events directly to the Integration Adapter Framework (IAF). To do this, specify the port of the IAF. By default, this is 16903.

## Options

The `engine_send` tool takes the following options:

Option	Description
<code>-h</code>   <code>--help</code>	Displays usage information. Optional.
<code>-n host</code>   <code>--hostname host</code>	Name of the host on which the event correlator to which you want to send events is running. Optional. The default is <code>localhost</code> . Non-ASCII characters are not allowed in host names.
<code>-p port</code>   <code>--port port</code>	Port on which the event correlator is listening. Optional. The default is 15903.
<code>-c channel</code>   <code>--channel channel</code>	For events for which a channel is not specified, this option designates the delivery channel. If a channel is not specified for an event and you do not specify this option, the event is delivered to the default channel, which is the empty string. All

Option	Description
	<p>public contexts receive events sent to the default channel. All queries receive events sent to the default channel.</p> <p>To send events to only running Apama queries, specify the <code>com.apama.queries</code> channel. See "Defining Queries" in <i>Developing Apama Applications</i>.</p>
<code>-l count   --loop count</code>	<p>Number of times to cycle through and send the input events. Optional. Replace <i>count</i> with one of the following values:</p> <ul style="list-style-type: none"> <li>■ 0 — Indicates that you want the <code>engine_send</code> tool to iterate through and send the input data once. This is the default.</li> <li>■ Any negative integer — Indicates that you want the <code>engine_send</code> tool to indefinitely cycle through and send the input events.</li> <li>■ Any positive integer — Indicates the number of times to cycle through and send the input events.</li> </ul> <p>The <code>engine_send</code> tool ignores this option if you specify it and the input is from <code>stdin</code>.</p>
<code>-v   --verbose</code>	Requests verbose output during execution. Optional.
<code>-u   --utf8</code>	Indicates that input files are in UTF-8 encoding. This specifies that the <code>engine_send</code> tool should not convert the input to any other encoding.
<code>-V   --version</code>	Displays version information for the <code>engine_send</code> tool. Optional.

## Operands

The `engine_send` tool takes the following operands:

Operand	Description
<code>[ file1 [ file2 ... ] ]</code>	Specify zero, one, or more files that contain event data. Each file you specify must comply with the event file format described in " <a href="#">Event file format</a> " on page 219. If you do not



Operand	Description
	specify any filenames, the <code>engine_send</code> tool takes input from <code>stdin</code> .

### Exit status

The `engine_send` tool returns the following exit values:

Value	Description
0	The events were sent successfully.
1	No connection to the event correlator was possible or the connection failed.
2	One or more other errors occurred while sending the events.

### Operating notes

To end an indefinite cycle of sending events, press CTRL+C in the window in which you invoked the `engine_send` tool.

You might want to indefinitely cycle through and send events in the following situations:

- In test environments. For example, you can use `engine_send` to simulate heartbeats. If you then kill the `engine_send` process, you can test your EPL code that detects when heartbeats stop.
- In production environments. For example, you can use the `engine_send` tool to initialize a large data table in the correlator.

### Text encoding

By default, the `engine_send` tool checks the environment variable or global setting that specifies the locale because this indicates the local character set. The `engine_send` tool then translates EPL text from the local character set to UTF-8. Consequently, it is important to correctly set the machine's locale.

However, some input files might start with a UTF-8 Byte Order Mark. The `engine_send` tool treats such input files as UTF-8 and does not do any translation. Alternatively, you can specify the `-u` option when you run the `engine_send` tool. This forces the tool to treat each input file as UTF-8.

## Receiving events from correlators

The `engine_receive` tool lets you connect to a running event correlator and receive events from it. Events received and displayed by the `engine_receive` tool are in Apama event format. This is identical to the format used to send events to the correlator with the `engine_send` tool. Consequently, it is possible to reuse the output of the `engine_receive` tool as input to the `engine_send` tool.

The executable for this tool is located in the `bin` directory of the Apama installation. Running the tool in the Apama Command Prompt (see ["Setting up the environment using the Apama Command Prompt" on page 26](#)) ensures that the environment variables are set correctly.

### Synopsis

To receive Apama-format events from an event correlator, run the following command:

```
engine_receive [ options ]
```

When you run this command with the `-h` option, the usage message for this command is shown.

### Description

The `engine_receive` tool receives events from an event correlator and writes them to `stdout` or to a file that you specify. The correlator output format is the same as that used for event input and is described in ["Event file format" on page 219](#).

You can specify one or more channels on which to listen for events from the correlator. The default is to receive all output events. For more information, see "Subscribing to channels" in *Developing Apama Applications*.

To view progress information during `engine_receive` execution, specify the `-v` option.

You can also use `engine_receive` to receive events emitted by the Integration Adapter Framework (IAF) directly. To do this, specify the port of the IAF. By default, this is 16903.

### Options

The `engine_receive` tool takes the following options:

Option	Description
<code>-h</code>   <code>--help</code>	Displays usage information. Optional.
<code>-n host</code>   <code>--hostname host</code>	Name of the host on which the event correlator is running. Optional. The default is <code>localhost</code> . Non-ASCII characters are not allowed in host names.

Option	Description
<code>-p port   --port port</code>	Port on which the event correlator is listening. Optional. The default is 15903.
<code>-c channel   --channel channel</code>	Named channel on which to listen for output events from the correlator. Optional. The default is to listen for all output events. You can specify the <code>-c</code> option multiple times to listen on multiple channels.
<code>-f file   --filename file</code>	Dumps all received events in the specified file. Optional. The default is to write the events to <code>stdout</code> .
<code>-s   --suppressBatch   --suppressbatch</code>	Omits <code>BATCH</code> timestamps from the output events. Optional. The default is to preserve <code>BATCH</code> timestamps in events.
<code>-z   --zeroAtFirstBatch   --zeroatfirstbatch</code>	Records the first received batch of events as being received at 0 milliseconds after the <code>engine_receive</code> tool was started. Optional. The default is that the first received batch of events is received at the number of milliseconds since <code>engine_receive</code> actually started.
<code>-C   --logChannels</code>	Specifies that you want <code>engine_receive</code> output to include the channel that an event arrives on. If you then use the <code>engine_receive</code> output as input to <code>engine_send</code> , events are delivered back to the same-named channels. See <a href="#">"Event association with a channel" on page 222</a> .
<code>-r   --reconnect</code>	Automatically (re)connect to the server when available.
<code>-x   --qdisconnect</code>	Disconnect from the correlator if the <code>engine_receive</code> tool cannot keep up with the events from the correlator.
<code>-v   --verbose</code>	Requests verbose output during <code>engine_receive</code> execution. Optional.
<code>-u   --utf8</code>	Indicates that received event files are in UTF-8 encoding. This specifies that the <code>engine_receive</code> tool should not convert the input to any other encoding.

Option	Description
<code>-V   --version</code>	Displays version information for the <code>engine_receive</code> tool. Optional.
<code>-Xconfig file   --configFile file</code>	Specifies a special configuration file that the correlator uses to initialize its networking. Specify this option only as directed by Apama Technical Support to troubleshoot networking problems. For more information, see <a href="#">"Using the Apama component extended configuration file" on page 225</a> .

### Exit status

The `engine_receive` tool returns the following exit values:

Value	Description
0	All events were received successfully.
1	No connection to the event correlator was possible or the connection failed.
2	Other error(s) occurred while receiving events.

### Text encoding

The `engine_receive` tool translates all events it receives from UTF-8 into the current character locale. It is therefore important that you correctly set the machine's locale. To force the `engine_receive` tool to output events in UTF-8 encoding, specify the `-u` option.

## Watching correlator runtime status

The `engine_watch` tool lets you monitor the runtime operational status of a running event correlator. The executable for this tool is located in the `bin` directory of the Apama installation. Running the tool in the Apama Command Prompt (see ["Setting up the environment using the Apama Command Prompt" on page 26](#)) ensures that the environment variables are set correctly.

### Synopsis

To monitor the operation of an event correlator, run the following command:

```
engine_watch [ options ]
```

When you run this command with the `-h` option, the usage message for this command is shown.

### Description

This tool periodically polls an event correlator for status information, writing the standard status messages to `stdout` (see the status messages listed further below). When you also specify the `-a` option, any user-defined status values are appended to the standard status messages. For additional progress information, use the `-v` option.

### Options

The `engine_watch` tool takes the following options:

Option	Description
<code>-h   --help</code>	Displays usage information. Optional.
<code>-n host   --hostname host</code>	Name of the host on which the event correlator is running. The default is <code>localhost</code> . Non-ASCII characters are not allowed in host names.
<code>-p port   --port port</code>	Port on which the event correlator is listening. Optional. The default is <code>15903</code> .
<code>-i ms   --interval ms</code>	Specifies the poll interval in milliseconds. Optional. The default is <code>1000</code> .
<code>-f filename   --filename filename</code>	Writes status output to the named file. Optional. The default is to send status information to <code>stdout</code> .
<code>-r   --raw</code>	Indicates that you want raw output format, which is more suitable for machine parsing. Raw output format consists of a single line for each status message. Each line is a comma-separated list of status numbers. This format can be useful in a test environment.  If you do not specify that you want raw output format, the default is a multi-line, human-readable format for each status message.
<code>-a   --all</code>	Outputs all user-defined status values after the standard status messages. Optional. The default is to output only the standard status messages.

Option	Description
<code>-t   --title</code>	If you also specify the <code>--raw</code> option, you can specify the <code>--title</code> option so that the output contains headers that make it easy to identify the columns.
<code>-o   --once</code>	Outputs one set of status information and then quits. Optional. The default is to indefinitely return status information at the specified poll interval.
<code>-v   --verbose</code>	Displays process names and versions in addition to status information. Optional. The default is to display only status information.
<code>-V   --version</code>	Displays version information for the <code>engine_watch</code> tool. Optional. The default is that the tool does not output this information.

### Standard status messages

The `engine_watch` tool returns the following standard status messages:

Status message	Meaning
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 monitors	The number of EPL 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, or it has no listeners or streams left, or it causes a runtime error.
Number of sub-monitors	The number of EPL monitor instances across all contexts in the correlator. In monitors, <code>spawn</code> actions create monitor instances. This figure changes upwards and downwards as monitor instances are spawned, killed or just expire.
Number of contexts	The number of contexts in the correlator. This includes the main context plus any user-defined contexts.

Status message	Meaning
Number of Java applications	The number of Java applications loaded in the correlator. Java applications do not expire, so this value only decreases when you explicitly unload a Java application.
Number of listeners	<p>This is the sum of listeners in all contexts. This includes each <code>on</code> statement and each stream source template, for example, <code>all Tick(symbol="APMA")</code> in the following:</p> <pre>stream&lt;Tick&gt; ticks := all Tick(symbol="APMA");</pre>
Number of sub-listeners	The number of sub-listeners that have been created by listeners across all contexts. A stream source template always has exactly one sub-listener. An <code>on</code> statement can have multiple sub-listeners.
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 total number of events waiting to be processed on all input queues. The main context has its own input queue and any user-defined contexts each have an input queue. This includes private contexts as well as public contexts.
Events received	<p>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.</p> <p>Note that if an event is on an input queue, it has been received but not processed.</p>
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 and streams that were waiting for it have been triggered, or when it has been determined that there are no listeners for the event.

Status message	Meaning
Events on internal queue	Total number of routed events waiting to be processed across all contexts. The internal routing queue in each context is a high priority queue for events that you internally routed with the <code>route</code> instruction in EPL. The correlator always processes events on the internal queue before any events on the normal input queue.
Events routed internally	The total number of events ever routed internally to the internal queues on this 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 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 (also called "receivers" here) registered with the correlator. Receivers 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 receivers.
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 receivers. For example, suppose the correlator created 10 output events and sent each event to two receivers. 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.
Event rate over last interval (ev/s)	The number of events per second currently being processed by the correlator across all contexts.



Status message	Meaning
	This value is computed with every status refresh and is only an approximation.
Events on input context queues	The total number of events on all input context queues. This is the sum of the queue size of main context and any context marked as input by passing <code>true</code> as the second parameter to the context constructor.
Slowest context name	The context which is the slowest - most behind in time or number of events.
Slowest context queue size	The number of events on the slowest context's input queue.
Slowest receiver name	The receiver with the largest number of incoming events waiting to be processed.
Slowest receiver queue size	For the receiver with the largest number of incoming events waiting to be processed, this is the number of events that are waiting.

### Types of receivers

The term "receiver" which is used in the above table refers to any of the following:

- EPL, Java or C++ plug-ins using the `Correlator.subscribe` method.
- Connectivity plug-ins for "towards" transport events.
- JMS or UM connections sending events out of the correlator.
- Client library connections, including other correlators that have been connected with the `engine_connect`, `iaf` or `engine_receive` tools.

### Exit status

The `engine_watch` tool returns the following exit values:

Value	Description
0	All status requests were processed successfully.
1	No connection to the event correlator was possible or the connection failed.

Value	Description
2	Other error(s) occurred while requesting/processing status.

## Inspecting correlator state

The `engine_inspect` tool lets you inspect the state of a running event correlator. This means you can review the applications loaded and running on a correlator. The executable for this tool is located in the `bin` directory of the Apama installation. Running the tool in the Apama Command Prompt (see ["Setting up the environment using the Apama Command Prompt" on page 26](#)) ensures that the environment variables are set correctly.

### Synopsis

To inspect applications on a running event correlator, run the following command:

```
engine_inspect [ options ]
```

When you run this command with the `-h` option, the usage message for this command is shown.

### Description

The `engine_inspect` tool retrieves state information from a running event correlator and sends it to `stdout`. By default, the tool outputs information on the monitors, JMon applications, event types and container types currently injected in an event correlator.

You can filter this list by specifying command-line options. When you specify one or more of the `-m`, `-j`, `-e`, `-t`, `-x`, `-P`, or `-R` options, the `engine_inspect` tool displays only the information indicated by the option(s) you specify. See the table below for more information on these options.

### Options

The `engine_inspect` tool takes the following options:

Option	Description
<code>-m</code>   <code>--monitors</code>	Displays the names of all EPL monitors in the event correlator and the number of sub-monitors each monitor has spawned.
<code>-j</code>   <code>--java</code>	Displays the names of all JMon applications in the event correlator and the number of event listeners each JMon application has created.

Option	Description
<code>-e   --events</code>	<p>Displays the names of all event types in the correlator and the number of templates of each type, as defined in listener specifications. This includes each event template in an <code>on</code> statement and each stream source template, for example, <code>stream&lt;A&gt; := all A()</code>.</p> <p>For more information about event types and listeners, see "Introduction to Apama Event Processing Language" in <i>Developing Apama Applications</i>.</p>
<code>-t   --timers</code>	<p>Displays the current EPL timers active within the system. The four types of timers which may be displayed here are <code>wait</code>, <code>within</code>, <code>at</code>, and <code>stream</code>. The <code>stream</code> timers are those set up to support the operation of a stream network.</p>
<code>-x   --contexts</code>	<p>Displays the names of any user-defined contexts, how many monitor instances are running in each context, what channels each context is subscribed to, and how many entries are on each context's input queue.</p>
<code>-a   -- aggregates</code>	<p>Displays a list of the custom (user-defined) aggregate functions that have been injected. You use aggregate functions in stream queries. Apama built-in aggregate functions are not listed.</p>
<code>-P   -- pluginReceivers</code>	<p>Displays the names of any plug-in receivers, the channels the plug-in is subscribed to, and the number of items on the plug-in's input queue. A plug-in receiver is a correlator plug-in that is subscribed to one or more channels.</p>
<code>-R   --receivers</code>	<p>Displays the names of any external receivers, each receiver's address, the channels each receiver is subscribed to, and the number of entries on each receiver's output queue.</p>
<code>-r   --raw</code>	<p>Indicates that you want raw output, which is more suitable for machine parsing. Raw output provides the name of each entity in the correlator followed by the number of instances associated with that entity. For a monitor, you get the number of its monitor instances. For a JMon application, you get the number of its listeners. For an event type, you get the number of its templates. For example:</p> <pre>com.apama.samples.stockwatch.StockWatch 1</pre>

Option	Description
	Tick 1
<code>-h   --help</code>	Displays usage information.
<code>-n host   --hostname host</code>	Name of the host on which the event correlator is running. The default is <code>localhost</code> . Non-ASCII characters are not allowed in host names.
<code>-p port   --port port</code>	Port on which the event correlator is listening. The default is 15903.
<code>-v   --verbose</code>	Displays process names and versions in addition to application information. Optional. The default is to display only application information.
<code>-V   --version</code>	Displays version information for the <code>engine_inspect</code> tool.

### Exit status

The `engine_inspect` tool returns the following exit values:

Value	Description
0	All status requests were processed successfully.
1	No connection to the event correlator was possible or the connection failed.
2	Other error(s) occurred while requesting/processing status.

## Shutting down and managing components

All Apama components (correlator, IAF, Sentinel Agent, dashboard data server, and dashboard display server) implement an interface with which they can be asked to shut themselves down, provide their process ID, and respond to communication checks.

For historical reasons, there are several tools that all do the same thing. You can use any of these tools to manage any component:

- `engine_management`
- `component_management`

### ■ `iaf_management`

When managing a correlator, the recommendation is to use the `engine_management` tool, which provides some additional correlator-specific options that are not available in the other tools. The only other differences in behavior among these tools are:

- `engine_management` and `component_management` default to the local correlator port (15903).
- `iaf_management` defaults to the default IAF port (16903).

The executable for the `engine_management` tool is located in the `bin` directory of the Apama installation. Running the tool in the Apama Command Prompt (see ["Setting up the environment using the Apama Command Prompt" on page 26](#)) ensures that the environment variables are set correctly.

### Synopsis

To use the event correlator's management tool, run the following command:

```
engine_management [ options ]
```

When you run this command with the `-h` option, the usage message for this command is shown.

### Description

Use the `engine_management` tool to connect to a running component. Once connected, the tool can shut down the component or return information about the component. The `engine_management` tool can connect to any of the following types of components:

- Event correlator
- Adapter
- Sentinel Agent process
- Dashboard data server and dashboard display server (using the management port, and not the data port)

If you want to use the dedicated `dashboard_management` tool, see ["Managing and stopping the Data Server and Display Server" in \*Building and Using Dashboards\*](#).

The `engine_management` tool sends output to `stdout`.

### Options

The `engine_management` tool takes the following options. These options are also available for the `component_management` and `iaf_management` tools. Keep in mind that all of these tools use different ports (see above). To obtain all information for a particular component, specify the `-a` option. All options are optional.

Option	Description
<code>-V   --version</code>	Displays version information for the <code>engine_management</code> tool.
<code>-h   --help</code>	Display usage information.
<code>-v   --verbose</code>	Displays information in a more verbose manner. For example, when you specify the <code>-v</code> option, the <code>engine_management</code> tool displays status messages that indicate that it is trying to connect to the component, has connected to the component, is disconnecting, is disconnected, and so on. If you are having trouble obtaining the information you want, specify the <code>-v</code> option to help determine where the problem is.
<code>-n host   --hostname host</code>	Name of the host on which the component is running. The default is <code>localhost</code> . Non-ASCII characters are not allowed in host names.
<code>-p port   --port port</code>	Port on which the component you want to connect to is listening. The default is 15903.
<code>-w   --wait</code>	Instructs the <code>engine_management</code> tool to wait for the component to start and be in a state that is ready to receive EPL files. This option is similar to the <code>-W</code> option, except that this option (the <code>-w</code> option) instructs the tool to wait forever. The <code>-W</code> option lets you specify how many seconds to wait. See the information for the <code>-W</code> option for an example.
<code>-W num   --waitFor num</code>	Instructs the <code>engine_management</code> tool to wait <code>num</code> seconds for the component to start and be in a state that is ready to receive EPL files. If the component is not ready to receive EPL files before the specified number of seconds has elapsed,

Option	Description
	<p>the <code>engine_management</code> tool terminates with an exit code of 1.</p> <p>This option is most useful in scripts, when the component you want to operate on has not yet started. For example, suppose a script specifies the following commands:</p> <pre>correlator.exe options engine_inject some_EPL_files</pre> <p>It can sometimes take a few seconds for a component to start, and this number of seconds is not always exactly predictable. If the <code>engine_inject</code> tool runs before the correlator is ready to receive EPL files, the <code>engine_inject</code> tool fails. To avoid this for a local correlator that is listening on the default port, insert the following command between these commands:</p> <pre>engine_management -W 10</pre> <p>This lets the <code>engine_management</code> tool wait for up to 10 seconds for the correlator's management interface to be available. To set an appropriate wait time for your application, monitor your application's performance and adjust as needed.</p>
-N   --getname	<p>Displays the name of the component. For example, when you start a correlator, you can give it a name with the <code>-N</code> option. This is the name that the <code>engine_management</code> tool returns. If you do not assign a name to a correlator when you start it, the default name is <code>correlator</code>.</p>
-T   --gettype	<p>Displays the type of the component that the <code>engine_management</code> tool connects to. The returned value is one of the following: <code>correlator</code>, <code>iaf</code>, <code>sentinel_agent</code>. If you see that a port is in use, you can specify this option to determine the type of component that is using that port.</p>
-M   --getuptime	<p>Gets the uptime of the component in milliseconds. This can be useful if you wish to track when and for how long a</p>

Option	Description
	particular component has been running for.
<code>-Vm   --getvmemory</code>	<p>Gets the virtual memory usage of the component in megabytes. This can be useful if you wish to measure the virtual memory usage of a component, for example, to identify possible memory leaks.</p> <p>For the Java-based dashboard data server and display server, the virtual memory value returned is the total of the heap and non-heap "used" memory, as given by the <code>java.lang.management.MemoryMXBean</code> class.</p>
<code>-Pm   --getpmemory</code>	<p>Gets the physical memory usage of the component in megabytes. This can be useful if you wish to measure the physical memory usage of a component, for example, to identify possible memory leaks.</p> <p>For the Java-based dashboard data server and display server, the physical memory value returned is the total of the heap and non-heap "committed" memory, as given by the <code>java.lang.management.MemoryMXBean</code> class.</p>
<code>-Y   --getphysical</code>	Displays the physical ID of the component. This can be useful if you are looking at status log information that identifies components by their physical IDs.
<code>-L   --getlogical</code>	Displays the logical ID of the component. This can be useful if you are looking at status log information that identifies components by their logical IDs.
<code>-O   --getloglevel</code>	<p>Displays the log level of the component. The returned value is one of the following: TRACE, DEBUG, INFO, WARN, ERROR, CRIT, FATAL, or OFF.</p>



Option	Description
-C   --getversion	Displays the version of the component. For example, when the tool connects to a correlator, it displays the version of the correlator software that is running.
-R   --getproduct	Displays the product version of the component. For example, when the tool connects to a correlator, it displays the version of the UNIX software that is running.
-B   --getbuild	Displays the build number of the component. This information is helpful if you need technical support. It indicates the exact software contained by the component you connected to.
-F   --getplatform	Displays the build platform of the component. This information is helpful if you need technical support. It indicates the set of libraries required by the component you connected to.
-P   --getpid	Displays the process ID of the correlator you are connecting to. This can be useful if you are looking at log information that identifies components by their process ID.
-H   --gethostname	Displays the host name of the component. When debugging connectivity issues, this option is helpful for obtaining the host name of a component that is running behind a proxy or on a multihomed system.
-U   --getusername	Displays the user name of the component. On a multiuser machine, this is useful for determining who owns a component.
-D   --getdirectory	Displays the working (current) directory of the component. This can be helpful if a plug-in writes a file in a component's working directory.

Option	Description
<code>-E   --getport</code>	Displays the port of the component.
<code>-c   --getconnections</code>	This option is for use by technical support. It displays all the connections to the component.
<code>-a   --getall</code>	Displays all information for the component.
<code>-xs id:id reason   --disconnectsender id:id reason</code>	Disconnects the sender that has the physical ID you specify. If you specify a reason, the <code>engine_management</code> tool sends the reason to the correlator. The correlator then logs the message, sends the reason to the sender, and disconnects the sender. You can specify the component ID as <i>physical_ID/logical_ID</i> .
<code>-xr id:id reason   --disconnectreceiver id:id reason</code>	Disconnects the receiver that has the physical ID you specify. If you specify a reason, the <code>engine_management</code> tool sends the reason to the correlator. The correlator then logs the message, sends the reason to the receiver, and disconnects the receiver. You can specify the component ID as <i>physical_ID/logical_ID</i> .
<code>-I category   --getinfo category</code>	This option is for use by technical support. It displays component-specific information for the specified category.
<code>-d   --deepping</code>	Ping the component. This confirms that the component process is running and acknowledging communications.
<code>-l level   --setloglevel level</code>	Sets the amount of information that the component logs. In order of decreasing verbosity, you can specify <code>TRACE</code> , <code>DEBUG</code> , <code>INFO</code> , <code>WARN</code> , <code>ERROR</code> , <code>FATAL</code> , <code>CRIT</code> , or <code>OFF</code> .
<code>-r type arg*   --dorequest type arg*</code>	This option sends a component-specific request. For example: <code>engine_management -r cpuProfile frequency</code> . This returns the profiling frequency in Hertz.

Option	Description
	<p>The following request types are available:</p>
■	<p><code>applicationEventLogging</code> — Sends detailed application information to the correlator log file. See <a href="#">"Viewing garbage collection information" on page 182</a>.</p>
■	<p><code>codeCoverage</code> — Lets you check which lines in an EPL file have been executed. See <a href="#">"Recording code coverage information" on page 210</a>.</p>
■	<p><code>flushAllQueues</code> — Sends a request into the correlator that waits until every event/injection sent or enqueued to a context before the <code>flushAllQueues</code> request started has been processed, and every event emitted as a result of those events has been acknowledged. This may block if a slow receiver is connected to the correlator. Events enqueued to a context after the request has started may or may not be processed — thus if you want to see the results of one context enqueueing to a second, which enqueues to a third, you should execute <code>engine_management -r flushAllQueues</code> three times, to ensure it has been processed by each context. This does not change the behavior of the correlator (the correlator will always flush all queues as soon as it is able to), it just waits for events currently on input queues to complete.</p>
■	<p><code>eplMemoryProfileOverview</code> — Returns information on all the monitors in the correlator. See <a href="#">"Using the EPL memory profiler" on page 182</a>.</p>
■	<p><code>eplMemoryProfileMonitorInstanceDetail</code> — Returns monitor instance details. See <a href="#">"Using the EPL memory profiler" on page 182</a>.</p>
■	<p><code>eplMemoryProfileMonitorDetail</code> — Returns aggregated monitor instance details. See <a href="#">"Using the EPL memory profiler" on page 182</a>.</p>

Option	Description
	<ul style="list-style-type: none"> <li>■ <code>cpuProfile</code> — Lets you profile Apama EPL applications. See <a href="#">"Using the CPU profiler" on page 189</a>.</li> <li>■ <code>rotateLogs</code> — See the description for the corresponding option below.</li> <li>■ <code>setApplicationLogFile</code> — See the description for the corresponding option below.</li> <li>■ <code>setApplicationLogLevel</code> — See the description for the corresponding option below.</li> <li>■ <code>setLogFile</code> — See the description for the corresponding option below.</li> <li>■ <code>verbosegc</code> — Enables logging of garbage collection events. See <a href="#">"Viewing garbage collection information" on page 182</a>.</li> </ul> <p>Certain other requests for this option are available for use by Apama technical support.</p>
<code>-s why   --shutdown why</code>	Instructs the component to shut down and specifies a message that indicates the reason for termination. The component inserts the string you specify in its status log file with a <code>CRIT</code> flag, and then shuts down.
<code>--setLogFile path</code>	Instructs the component to close the log file it is using and open a new log file with the name you specify. See <a href="#">"Rotating specified log files" on page 198</a> .
<code>--reopenLog</code>	Reopens the log file of the component.

The following correlator-specific options, which replicate `--doRequest` commands, are only available for the `engine_management` tool.

Option	Description
<code>--rotateLogs</code>	Rotates all the log files. See <a href="#">"Rotating all log files" on page 198</a> .

Option	Description
<code>--setApplicationLogLevel</code> <code>[package=] level</code>	Sets the log level for application log messages (global or per-package). For more information on how to set, get and unset the log level, see <a href="#">"Setting logging attributes for packages, monitors and events" on page 194</a> .
<code>--setApplicationLogFile</code> <code>[package=] path</code>	Sets the log file for application log messages (global or per-package).  For more information on how to set, get and unset the log file, see <a href="#">"Setting logging attributes for packages, monitors and events" on page 194</a> .
<code>--unsetApplicationLogLevel</code> <code>package</code>	Unsets the application log level for this package.
<code>--unsetApplicationLogFile</code> <code>package</code>	Unsets the application log file for this package.
<code>--unsetRootApplicationLogLevel</code>	Unsets the root application log level.
<code>--unsetRootApplicationLogFile</code>	Unsets the root application log file.
<code>--getApplicationLogLevel</code> <code>package</code>	Displays the application log level for this package.
<code>--getApplicationLogFile</code> <code>package</code>	Displays the application log file for this package.
<code>--getRootApplicationLogLevel</code>	Displays the root application log level.
<code>--getRootApplicationLogFile</code>	Displays the root application log file.

### Exit values

The `engine_management` tool returns the following exit values:

Value	Description
0	All status requests were processed successfully.

Value	Description
1	Indicates one of the following: <ul style="list-style-type: none"> <li>■ No connection to the specified component was possible.</li> <li>■ The connection failed.</li> <li>■ You specified the <code>waitFor</code> option and the specified time elapsed without the component starting.</li> </ul>
2	One or more errors occurred while requesting/processing status.
3	Deep ping failed.

## Viewing garbage collection information

A handy way to view garbage collection information is to execute the following command:

```
engine_management -r verbosegc on
```

This command enables logging of garbage collection events, and is particularly useful in production environments. The additional garbage collection information goes to the correlator log. To disable logging of garbage collection information, execute the following:

```
engine_management -r verbosegc off
```

These commands provide an alternative to the following command, which provides a great deal of detailed output in addition to garbage collection information. Again, this output goes to the correlator log.

```
engine_management -r applicationEventLogging on
```

To turn this off:

```
engine_management -r applicationEventLogging off
```

## Using the EPL memory profiler

You use the EPL memory profiler to display information on monitors and monitor instances.

The EPL memory profiler is invoked using the `-r` (or `--dorequest`) option of the `engine_management` tool, followed by a request. Several requests are available for the EPL memory profiler, which are described below.

**Important:** Do not use these requests on latency-sensitive applications. You should use them routinely only when developing or debugging.

When a request is issued, the correlator execution is momentarily paused to gather statistics.

The information that is returned for a request can be viewed directly (for example, in the Apama Command Prompt), or it can be written to a comma-separated values (CSV) file which can easily be viewed in tabular form using a tool such as Microsoft Excel.

**Note:** All byte counts returned by a request are approximate values. In theory, they will add up to total memory usage of all objects. Certain objects such as strings, however, are shared. Therefore, the memory usage is always approximate and not an exact value.

The values returned for the number of bytes and the number of EPL objects also include EPL objects that are no longer being used, and have not yet been garbage-collected. Therefore, the values will never be precise unless you are lucky enough to make this request just after garbage collection has run. See also "Garbage collection" in *Developing Apama Applications*.

The size of event expressions, including internal data structures associated with them, is excluded (and is typically small).

Each request returns the following string, in addition to the column headers described below:

```
"Version:version, Snapshot time:time, Component ID:id, Host:host-name,
Port:port, EPL memory:bytes"
```

String element	Output
Version:version	Information on the correlator version.
Snapshot time:time	Time at which the EPL memory profiler has taken the snapshot. This is the date in milliseconds. The date "1446716459541", for example, translates into "Thu Nov 05 2015 09:40:59" in UTC time.
Component ID:id	Correlator component ID.
Host:host-name	Name of the host on which the correlator is running.
Port:port	Port number on the above host.
EPL memory:bytes	Total memory used by the EPL types in the correlator.

### Returning information on all monitors

The following command returns information on all the monitors in the correlator:

```
engine_management -r eplMemoryProfileOverview
```

This request does not take arguments. If arguments are passed, they are ignored.

The output shows the following information in the following order:

Column header	Information shown in this column
Monitor	The name of the monitor.
Monitor instances	The number of monitor instances.
EPL objects	The number of EPL objects created by the monitor instances (for example, dictionaries, events, sequences, and so on).
Listeners	The number of active listeners.
Bytes	The approximate number of bytes used by EPL objects created by the monitor instances.
Overhead bytes	The approximate number of bytes covering miscellaneous internals that the correlator maintains for book-keeping per monitor instance.

### Returning monitor instance details

The following command returns information for all EPL types across the monitor instances of a specific monitor in the correlator:

```
engine_management -r eplMemoryProfileMonitorInstanceDetail monitor-name
```

where *monitor-name* is the name of the monitor. You can also specify `all` to list the instance details of all monitors, sorted by the monitor name.

The output shows the following information in the following order:

Column header	Information shown in this column
Monitor	The name of the monitor.
Persistent	<code>true</code> if the monitor is persistent. <code>false</code> it is not persistent.
EPL type	The type of the EPL object (see "Types" in the "EPL Reference", which is part of <i>Developing Apama Applications</i> ) and also any active listeners. The output shows one entry for each listener. For example, if there is a monitor with one instance, and which has 2 listeners where each listener



Column header	Information shown in this column
	has 10 active instances, then the output will contain 2 rows. The number of EPL objects will then be 10 for each row.
Context name	The name of the context.
Context ID	The ID of the context.
Monitor instance ID	The ID of the monitor instance.
EPL objects	The number of EPL objects created by the monitor instances (for example, dictionaries, events, sequences, and so on).
Bytes	The approximate number of bytes used by EPL objects created by the monitor instances.

The output for the context is a combination of EPL type and monitor instance. For example, if there are 10 monitor instances where each instance has lots of objects of 3 different types, then the output will have 30 rows.

Unlike other EPL objects which belong to a single monitor instance, some strings are shared between several monitor instances. When a string is only used by a single monitor instance, it is shown like any other object in the output of the request, that is, with an EPL type of "string". However, if the same string is shared between multiple monitor instances, then each monitor or monitor instance that is using it will show the EPL type as "string (shared)". This is a performance optimization which avoids unnecessary copying. For example, a string may be shared in the following cases:

- When a monitor containing a string spawns to another monitor instance.
- When a monitor has a string that it sends inside an event to a monitor in another context.
- When an input event containing strings is received by multiple monitor instances which then store these strings.

One of the implications of sharing is double-counting, for both the number of EPL objects and the number of bytes. If multiple monitor instances refer to the same shared strings, the output of the request will include these numbers against each monitor instance separately. However, the duplication is eliminated when object sizes are summed up for the "EPL memory" value, so it may end up being notably lower than the sum of the "Bytes" in each row.

**Note:** For reference types (such as sequence and dictionaries), the size is not reflected in the object referencing it. Instead, the size is associated with the actual object which is referenced. For example, if an event references/contains

a sequence, the size of the sequence has no effect on the byte count of that event.

### Returning aggregated monitor instance details

The following command is similar to `eplMemoryProfileMonitorInstanceDetail`, except that it aggregates the object count and size from each monitor instance, displaying data per monitor rather than per monitor instance.

```
engine_management -r eplMemoryProfileMonitorDetail monitor-name
```

where `monitor-name` is the name of the monitor. You can also specify `all` to list all monitors, sorted by the monitor name.

The output shows the following information in the following order:

Column header	Information shown in this column
Monitor	The name of the monitor.
Persistent	<code>true</code> if the monitor is persistent. <code>false</code> if it is not persistent.
EPL type	The type of the EPL object (see "Types" in the "EPL Reference", which is part of <i>Developing Apama Applications</i> ) and also any active listeners. The output shows one entry for each listener. For example, if there is a monitor with one instance, and which has 2 listeners where each listener has 10 active instances, then the output will contain 2 rows. The number of EPL objects will then be 10 for each row.
EPL objects	The number of EPL objects created by the monitor instances (for example, dictionaries, events, sequences, and so on).
Bytes	The approximate number of bytes used by EPL objects created by the monitor instances.

This request also takes account of shared strings. See the description of the `eplMemoryProfileMonitorInstanceDetail` request for details.

**Note:** In the case of reference types (such as sequence and dictionaries), the size is not reflected in the object referencing it. Instead, the size is associated with the actual object which is referenced. For example, if an event references/contains a sequence, the size of the sequence has no effect on the byte count of that event.

### Visualizing the EPL memory profiler information in Microsoft Excel

You can write the output from each of the above requests to a comma-separated values (CSV) file which can easily be viewed in tabular form using a tool such as Microsoft Excel.

The example below shows how to visualize the output of the `eplMemoryProfileMonitorInstanceDetail` request in Microsoft Excel using a pivot table.

1. Save the output of the request in a CSV file and open this file with Microsoft Excel.
2. Create a new **PivotTable** in Microsoft Excel, and in the resulting dialog select the range of data for the pivot table (ideally, the information for the entire range is already provided in the corresponding text box).
3. For this example, add the following fields to the report:

- Monitor
- EPL type
- Context name
- EPL objects
- Bytes

If required, you can also add the following fields:

- Monitor instance ID
- Context ID

The monitor instance ID is helpful if multiple monitor instances exist within the same context.

After you have added the fields, you can see the following in the table:

- row labels which include the monitor name, the context name and the EPL type, and
- columns which sum up the number of EPL objects and the approximate number of bytes.

For example:

3	Row Labels	Sum of	EPL objec	Sum of	Bytes
4	com.apama.primitive.PrimitiveMonitor		38		25600
5	main		14		1952
6	com.apama.primitive.Configure		1		120
7	com.apama.primitive.PrimitiveMonitor.Global Variables		1		136
8	context		4		384
9	Listener - PrimitiveMonitor.mon:29		1		192
10	Listener - PrimitiveMonitor.mon:31		4		800
11	Listener - PrimitiveMonitor.mon:40		1		200
12	sequence(float)		1		112
13	string		1		8
14	Primitive-Context1		6		4840
15	com.apama.primitive.Configure		1		120
16	com.apama.primitive.PrimitiveMonitor.Global Variables		1		120
17	Listener - PrimitiveMonitor.mon:48		1		192
18	Listener - PrimitiveMonitor.mon:52		1		192
19	sequence(float)		1		4208
20	string		1		8
21	Primitive-Context2		6		17128
22	com.apama.primitive.Configure		1		120
23	com.apama.primitive.PrimitiveMonitor.Global Variables		1		120
24	Listener - PrimitiveMonitor.mon:48		1		192
25	Listener - PrimitiveMonitor.mon:52		1		192
26	sequence(float)		1		16496
27	string		1		8
28	Primitive-Context3		6		808
29	com.apama.primitive.Configure		1		120
30	com.apama.primitive.PrimitiveMonitor.Global Variables		1		120
31	Listener - PrimitiveMonitor.mon:48		1		192
32	Listener - PrimitiveMonitor.mon:52		1		192
33	sequence(float)		1		176
34	string		1		8
35	Primitive-Context4		6		872
36	com.apama.primitive.Configure		1		120
37	com.apama.primitive.PrimitiveMonitor.Global Variables		1		120
38	Listener - PrimitiveMonitor.mon:48		1		192
39	Listener - PrimitiveMonitor.mon:52		1		192
40	sequence(float)		1		240
41	string		1		8
42	com.apama.reference.ReferenceTypeMonitor		12139		1471080
43	main		15		2280
44	com.apama.reference.Configure		1		120
45	com.apama.reference.ReferenceTypeMonitor.Global Variables		1		136

- If you want to get an overview of the context level in the row labels, just drag the **Context name** label above the **Monitor** label as shown in the example below, and then check the changes in the report:

Drag fields between areas below:

Report Filter	Column Labels
	Σ Values
Row Labels	Σ Values
Context na...	Sum of EPL...
Monitor	Sum of Bytes
EPL type	

- Similarly, if you want to see how the EPL objects are distributed over the different contexts and monitors, just move the **EPL type** to the very top of the row labels, followed by the **Monitor** and **Context name** labels.
- Once the data is shown as wanted in the table, you can conditionally format the table to highlight individual columns, for example, to show high values or values that are above the threshold or above the average. Detailed information on how to do this can be found in the Excel help.

A basic use case is to highlight the values for the object count and byte count that are above the average. To do so, select the **Sum of EPL objects** column and then choose the following command: **Conditional Formatting > Top/Bottom Rules > Above Average**. You can then select a formatting option from a dialog, for example, red text. As a result, all values in the cells of the **Sum of EPL objects** column that are above the average are shown in red. If you want, you can do the same for the **Sum of Bytes** column.

You can also use additional conditional formatting (for example, color scales) to highlight the cells with values above the average.

## Using the CPU profiler

Using the CPU profiler, you can profile applications written with EPL. Data collected in the profiler allows you to identify possible bottlenecks in an EPL application. When testing an application, or after you deploy an application, you might find it handy to write a script that includes obtaining profile information. The CPU profiler that is described here allows you to obtain profile information without the overhead of Software AG Designer (see also "Profiling EPL Applications" in *Using Apama with Software AG Designer*).

The CPU profiler is invoked using the `-r` (or `--dorequest`) option of the `engine_management` tool:

```
engine_management -r cpuProfile argument
```

where *argument* can be one of the following:

Argument	Description
<code>on</code>	Starts to capture the state of all contexts in the correlator.
<code>off</code>	Stops capturing profile data.
<code>get</code>	Returns the samples collected since the correlator was started or since the profiler was reset. Returned data is in CSV (comma-separated values) format. A sample is the state of the correlator at the moment the profiler collects data.
<code>gettotal</code>	Returns totals for all contexts.
<code>reset</code>	Clears profiling samples collected.
<code>frequency</code>	Returns the profiling frequency in Hertz.

If a context is executing, it is typically in the EPL interpreter. However, it might also be doing something such as matching events or collecting garbage. For EPL execution, there is a call stack for each context. For the purposes of the profiler, there is one entry at the top for the monitor name, then comes the listener/`onload` action, and then any actions that is calling, and so on. The only action that the correlator is actually executing is at the bottom of the stack.

A context can be in one or two of the following states:

- **CPU.** The correlator is executing code in this context.
- **Runnable.** The correlator has work to do in this context, but it has been rescheduled because the correlator is executing code in another context.
- **Idle.** The correlator has no work to do in this context.
- **Non-Idle.** The correlator has work to do in this context. When a context is in this state, it is also in one other state: CPU, Plug-in, Blocked, or Runnable.
- **Plug-in.** The correlator is executing a plug-in in this context.
- **Blocked.** The correlator cannot make progress in this context. It is blocked because of a full queue. The full queue might be the correlator output queue (the context is trying to emit an event) or another context's input queue.

When the profiler takes a sample, it examines every context in the correlator. Every entry in each context's call stack results in addition or modification of a line in the profiler output. The Cumulative column is incremented for all samples, and one or more of the other columns is incremented for the lowest (deepest) call stack element according to what states the context is in.

When the correlator is not executing EPL code, there is only one element in the stack, for example, when the correlator is processing an event.

The profiler's resolution is to a EPL action. That is, the profiler does not distinguish between lines within an action. The line number in the output is the first line of the action that generates code. For example, variable declarations without initializers, and comments do not generate code, while statements, and declarations with initializers, do generate code. The profiler treats the body of a listener (the code the correlator executes when the listener fires) as an action with the name `::listenerAction::`.

If you want to profile parts of a single large action, you need to split the action into multiple actions in order to determine where time is spent. Remember that action calls have some cost, so that could skew the results.

The `cpuProfile get` or `cpuProfile gettotal` request returns samples to stdout as lines of comma-separated values.

Output is sorted by context and then by CPU time. For example:

```
Context ID,Context name,Location,Filename and line number,Cumulative time,
CPU time,Empty,Non-Idle,Idle,Runnable,Plug-in,Blocked,Total ticks:573
3,3,processor:processor::listenerAction::,create-state.mon:
50,556,293,0,556,0,0,263,0
```

In the above output, nearly all of the time of this context (3) is spent in the listener that starts on line 50 of `create-state.mon`. The time is spread between executing EPL code (293 samples) and executing a plug-in (263 samples). Each context spent similar amounts of time executing EPL and executing plug-ins but in different listeners (notice the different line numbers).

This output is intended to be imported to a spreadsheet, such as Microsoft Excel. If you do that, then the values in one sample (one row) provide the following information in the following order:

Column header	Information shown in this column
Context ID	ID of the context. A context ID is not present in data returned by <code>-r cpuProfile gettotal</code> .
Context name	Name of the context. A context name is not present in data returned by <code>-r cpuProfile gettotal</code> .
Location	<p>What the correlator is doing or where the correlator is executing code at the moment the sample was collected. The value is one of the following:</p> <ul style="list-style-type: none"> <li>■ <code>Monitor:monitor_name</code> — The top-level entry for the monitor.</li> <li>■ <code>monitor_name.code_owner.action_name</code> — For example, if monitor <code>monny</code> calls an action <code>act</code> on event <code>pkg.evie</code>, this location would be <code>monny.pkg.evie.act</code>.</li> </ul>

Column header	Information shown in this column
	<p>If a listener has been triggered, the action name is always <code>::listenerAction::</code>.</p> <ul style="list-style-type: none"> <li>■ <code>monitor_name.;GC</code> — Garbage collection.</li> <li>■ <code>Event:event_name</code> — Event matching or chastenment of an event of that type.</li> <li>■ <code>Idle</code> — Correlator has no work to do.</li> <li>■ There are other possible values that you might rarely see. They are self explanatory.</li> </ul>
Filename and line number	If the correlator is executing EPL code, indicates the filename and line number of the beginning of the action that is executing.
Cumulative time	Cumulative time indicates time spent in this location or in something that this location was calling (directly or indirectly). CPU time shows time spent in this location, not the actions it called.
CPU time	Number of samples in which the correlator is executing the location/action and is not in a plug-in (see Plug-in later in this table). CPU time is a subset of Cumulative time. It does not include time spent in the location(s) called by this location.
Empty	Number of samples in which the context was empty. An empty context should happen very rarely. A context might be empty if there is a race between getting the location and the state.
Non-Idle	Number of samples in which the context was at this row's location and not idle. Each sample in this count is also in the count for CPU time, Runnable, Plug-in, or Blocked.
Idle	<p>Number of samples in which the context was idle. This should correspond to a location of <code>Idle</code> or <code>Only just started profiling</code>, which means it is an unknown state.</p> <p>As with other cumulative counters, races can result in misleading results. For example, <code>Idle</code> in an action, but those are best ignored and should be small.</p>
Runnable	Number of samples in which the location was the lowest point on the call stack and the context was runnable. Runnable means it could have made progress, but the



Column header	Information shown in this column
	<p>scheduler determined that the correlator should run something else instead.</p> <p>When all rows contain 0 for this entry, it means that the correlator never (or very rarely) had to re-schedule one context to run another context. A non-zero value means this location was running for a long time, and it was suspended so that other contexts could run.</p>
Plug-in	Number of samples in which the location is executing a correlator plug-in.
Blocked	Number of samples in which the context was unable to make progress. For example, it was trying to emit an event but the correlator output queue was full, or it was trying to enqueue an event to a particular context but that context's input queue was full.

The `cpuProfile` request returns the following string, in addition to the column headers described above:

```
"Version:version, Snapshot time:time, Profile start time:time, Component
ID:id, Host:host-name, Port:port"
```

String element	Output
<code>Version:version</code>	Information on the correlator version.
<code>Snapshot time:time</code>	Time at which the CPU profiler has taken the snapshot. This is the date in milliseconds. The date "1446716459541", for example, translates into "Thu Nov 05 2015 09:40:59" in UTC time.
<code>Profile start time:time</code>	Time at which the CPU profiler has been started. This is the date in milliseconds.
<code>Component ID:id</code>	Correlator component ID.
<code>Host:host-name</code>	Name of the host on which the correlator is running.
<code>Port:port</code>	Port number on the above host.

## Setting logging attributes for packages, monitors and events

You can configure per-package logging in two ways:

- Statically, in the extended configuration file when starting the correlator. See ["Setting log files and log levels in an extended configuration file" on page 227](#).
- Dynamically, using the following `engine_management` options that are described here: `--setApplicationLogFile` and `--setApplicationLogLevel`.

In EPL code, you can specify `log` statements as a development or debug tool. By default, `log` statements that you specify in EPL send information to the correlator log file. If a log file was not specified when the correlator was started, and you have not executed the `engine_management` tool to associate a log file with the correlator, `log` statements send output to `stdout`.

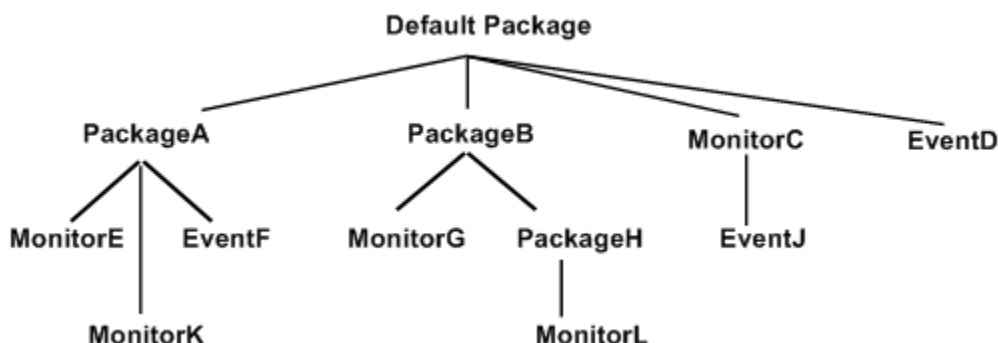
In place of this default behavior, you can specify different log files for individual packages, monitors and events. This can be helpful during development. For example, you can specify a separate log file for a package or monitor you are implementing, and direct log output from only your development code to that file.

Also, you can specify a particular log level for a package, monitor, or event. The settings of log files and log levels are independent of each other. That is, you can set only a log level for a particular package, monitor or event, or you can set only a log level for a particular element. The topics below provide information for managing individual log files and log levels.

See also ["Rotating the correlator log file" on page 197](#).

### Tree structure of packages, monitors, and events

Packages, monitors and events form a tree as illustrated in the figure below. For each node in the tree, you can specify a log file and/or a log level. Nodes for which you do not specify log settings inherit log settings from their parent node.



The root of the tree is the default package, which contains code that does not explicitly specify a package with the `package` statement. Specified packages are intermediate nodes. Packages can nest inside each other. Monitors and events in specified packages are leaf nodes. If you specify an event type in a monitor, that event is a leaf node and its containing monitor is an intermediate node.

For example, suppose you specify `packageA.log` as the log file for `packageA`. The `packageA.log` file receives output from log statements in `MonitorE` and `MonitorK`. If `EventF` contains any action members that specify log statements, output would go to the `packageA.log` file.

Now suppose that you set `ERROR` as the log level for the default package and you set `INFO` as the log level for `PackageB`. For log statements in `MonitorG`, `PackageH`, and `MonitorL`, the correlator compares the log statement's log level with `INFO`. For log statements in the rest of the tree, the correlator compares the log statement's log level with `ERROR`. For details, see the table in ["Managing application log levels" on page 195](#).

### Managing application log levels

To set the log level for a package, monitor or event, invoke the `engine_management` tool as follows:

```
engine_management --setApplicationLogLevel [node=] logLevel
```

Option	Description
<code>node</code>	Optionally, specify the name of a package, monitor or event. If you do not specify a node name, the tool sets the log level for the default package.
<code>logLevel</code>	Specify <code>OFF</code> , <code>CRIT</code> , <code>FATAL</code> , <code>ERROR</code> , <code>WARN</code> , <code>INFO</code> , <code>DEBUG</code> , or <code>TRACE</code> .

To obtain the log level for a particular node, invoke the tool as follows:

```
engine_management --getApplicationLogLevel [node]
```

If you do not specify a node, the tool returns the log level for the default package. To remove the log level for a node, so that it takes on the log level of its parent node, invoke the tool as follows. Again, if you do not specify a node, you remove the log level for the default package. The default package then takes on the log level in effect for the correlator. The default correlator log level is `INFO`.

```
engine_management --unsetApplicationLogLevel [node]
```

To manage the log level for an event that you define in a monitor, see ["Managing event logging attributes" on page 197](#).

After the correlator identifies the applicable log level, the log level itself determines whether the correlator sends the log statement output to the appropriate log file. The following table indicates which log level identifiers cause the correlator to send the log statement to the appropriate log file.

Log Level in Effect	Log Statements With These Identifiers Go to the Appropriate Log File	Log Statements With These Identifiers are Ignored
OFF	None	CRIT, FATAL, ERROR, WARN, INFO, DEBUG, TRACE
CRIT	CRIT	FATAL, ERROR, WARN, INFO, DEBUG, TRACE
FATAL	CRIT, FATAL	ERROR, WARN, INFO, DEBUG, TRACE
ERROR	CRIT, FATAL, ERROR	WARN, INFO, DEBUG, TRACE
WARN	CRIT, FATAL, ERROR, WARN	INFO, DEBUG, TRACE
INFO	CRIT, FATAL, ERROR, WARN, INFO	DEBUG, TRACE
DEBUG	CRIT, FATAL, ERROR, WARN, INFO, DEBUG	TRACE
TRACE	CRIT, FATAL, ERROR, WARN, INFO, DEBUG, TRACE	None

See also "Log levels determine results of log statements" in *Developing Apama Applications*.

### Managing application log files

To specify a log file for a package, monitor or event, invoke the `engine_management` tool as follows:

```
engine_management --setApplicationLogFile [node=] logFile
```

Option	Description
<code>node</code>	Optionally, specify the name of a package, monitor or event. If you do not specify a node name, the tool associates the log file with the default package.
<code>logFile</code>	Specify the path of the log file. You specify the name of an application log file in the same way that you specify the

Option	Description
	name of a correlator status file or input file. See <a href="#">"Specifying log filenames" on page 139</a> .

To obtain the path of the log file for a particular node, invoke the tool as follows:

```
engine_management --getApplicationLogFile [node]
```

If you do not specify a node, the tool returns the log file for the default package. To disassociate a log file from its node, so that the node uses the log file of its parent node, invoke the tool as follows. Again, if you do not specify a node, you disassociate the log file from the default package. The correlator log file is then in effect for the default package. If a log file has not been specified for the correlator, the default is `stdout`.

```
engine_management --unsetApplicationLogFile [node]
```

### Managing event logging attributes

If you specify an event type in a monitor, that event does not inherit the logging configuration from the enclosing monitor. It is expected that this will change in a future release. To explicitly set logging attributes for an event type defined in a monitor, invoke the `engine_management` tool and specify an unqualified event type name. Do not specify an enclosing scope, such as `com.apamax.myMonitor.NestedEventType`. For example:

```
engine_management --setApplicationLogFile NestedEventType=foo.log
engine_management --setApplicationLogLevel NestedEventType=DEBUG
```

## Rotating the correlator log file

Rotating the correlator log file refers to closing the status log file of a running correlator and opening a new status log file. This lets you archive log files and avoid log files that are too large to easily view.

Each site should decide on and implement its own correlator log rotation policy. You should consider

- How often to rotate log files
- How large a correlator log file can be
- What correlator log file naming conventions to use to organize log files.

There is a lot of useful header information in the log file being used when the correlator starts. If you need to provide log files to Apama technical support, you should be able to provide the log file that was in use when the correlator started, as well as any other log files that were in use before and when a problem occurred.

To rotate the correlator log file and also rotate any other log file the correlator is using (input log file, application log files), see ["Rotating all log files" on page 198](#).

To rotate only the correlator log file, see ["Rotating specified log files" on page 198](#).

## Rotating all log files

To invoke rotation of all log files that the correlator is using, you can do the following:

- Invoke the `engine_management` tool and specify `--rotateLogs`.

This rotates the correlator status log, the sentinel agent status log, the correlator input log if it is being generated, and any application logs that are being generated. When you invoke this management request then the correlator closes each log file it was using.

If the log filename specification declared `${START_TIME}`, `${ROTATION_TIME}` and/or `${ID}` then the correlator starts new log files with updated names according to the log filename specification, for example, if `${ID}` was specified then the ID portion of a log filename would be incremented by 1.

- In EPL, create a monitor that uses the Management interface correlator plug-in to trigger log rotation on a schedule. See *Using the Management interface in Deploying and Managing Apama Applications*.
- On UNIX only, you can write a `cron` job that periodically sends a `SIGHUP` signal to Apama processes.

The standard UNIX `SIGHUP` mechanism causes Apama processes to re-open their log files. If the log file names were specified with the `${ROTATION_TIME}` and/or `${ID}` then the re-opened files have names that contain the rotation time and/or the incremented ID.

If you want a log filename to always be the same and so did not declare `${START_TIME}`, `${ROTATION_TIME}` or `${ID}` in the log filename specification then the correlator starts new log files that have the same names as the log files it closed. On Windows, this would overwrite the closed log files so you must move the log files before invoking rotation. On UNIX, log files are appended to if the names are the same.

## Rotating specified log files

Run one of the following utilities to rotate a particular log file. On Windows, set up scheduled tasks that run the utilities. On UNIX, write a `cron` job that periodically runs the utilities. The behavior is the same on both Windows and UNIX, except as noted. The only way to rotate the correlator input log is to rotate all log files. See ["Rotating all log files" on page 198](#).

- The following command instructs the correlator to close its status log file and start using a status log file that has the name you specify. If the name of the file contains blanks, be sure enclose it in quotation marks.

```
engine_management --setLogFile log-filename
```

- The following command instructs the correlator to use the specified file as the log file for the specified node, which can be a package, monitor, or event. See also ["Setting logging attributes for packages, monitors and events" on page 194](#).

```
engine_management --setApplicationLogFile [node=]log-filename
```

If you use separate log files for particular packages, monitors, or events you might want to rotate those logs at the same time that you rotate the correlator status log. This keeps your Apama log files in sync with each other. See ["Rotating all log files" on page 198](#).

On Windows, when you rotate a log file you must ensure that the new log filename is different from the name of the log file that was in use. Apama takes care of this for you if you specify `${ROTATION_TIME}` and/or `${ID}` in the `log-filename` specification. If the name is not different, the old file is overwritten. If you want to use the same log filename then you must move the file before you rotate it.

On UNIX, a log file is never overwritten. If you rotate a log file and specify the same name then Apama appends messages to the content already there.

Apama does not support automatic log file rotation based on log file size.

The only way to rotate the correlator input log is to rotate all log files. See ["Rotating all log files" on page 198](#).

## Using the command-line debugger

The `engine_debug` tool lets you control execution of 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 executable for this tool is located in the `bin` directory of the Apama installation.

In general, this tool is expected to be most useful when you are ready to deploy your application or after deployment. During development, the interactive debugger in Software AG Designer will probably be most useful to you.

Before you run the `engine_debug` tool, specify the `-g` option when you start the correlator. Specification of this option disables some correlator optimizations. If you run the `engine_debug` tool and you did not specify the `-g` option when you started the correlator, the optimizations hinder the debugging process. For example, the correlator might simultaneously execute multiple statements over multiple lines even if you are using debugger commands to step through the program line by line.

### Synopsis

To debug applications on a running event correlator, run the following command:

```
engine_debug [ [ global-options ] [command [options]] ... ]
```

To obtain a usage message, run the command with the `help` option.

### Description

Debugging a running correlator has some effect on the other programs that connect to that correlator. While you pause a correlator, the expected behavior of connected components is as follows:

- Sending events to the correlator continues to put events on the input queue of each public context. However, since the input queues are not being drained, if an input queue fills up, this will block senders, including the `engine_send` tool and adapters.
- The correlator sends out any events on its output queue. When the output queue is empty, receivers no longer receive events; no contexts are sending events.
- Other inspections of the correlator proceed as normal. For example, `engine_watch`, `engine_management`, and profiling data.
- You can shut down the correlator.
- You can inject monitors while the correlator is stopped. They will not run any of the `onload()` or similar code until the correlator resumes, but the inject call should succeed.
- Java applications continue to run completely independently of whether the correlator is stopped.
- All other requests block until the correlator resumes processing. This includes dumping correlator state, loading, and changing debug or profiling state.

The `engine_debug` tool is stateless. Consequently, during debugging, you can have multiple concurrent connections to the same correlator.

### Debug commands

The ordering of arguments to `engine_debug` commands works as follows:

- All arguments before the first command apply to all commands in that command line. This is useful for setting the host and port if you are not using the local defaults.
- All arguments following a command apply to only that command and they override any applicable arguments specified before the first command.
- The arguments to a particular command can be in any order
- When there are multiple commands in a line, the debugger executes them in the order in which they are specified. Execution continues until either all complete, or one fails, which prevents execution of any subsequent commands.

The `engine_debug` tool takes the following commands as options:

Abbreviation	Command	Description
h [ <i>command</i> ]	help [ <i>command</i> ]	Displays a usage message. To obtain help for a particular <code>engine_debug</code> command, specify that command.



Abbreviation	Command	Description
p	status	Displays the current debugger state, and position if stopped.
ha	hashes	Lists injected files and their hashes.
si	stepinto	Steps into an action.
sot	stepout	Steps out of an action.
sov	stepover	Steps over an instruction.
r	run	Begins processing instructions.
b	stop	Stops processing instructions.
w [-to int]	wait [--timeout timeout]	Waits for the correlator to stop processing instructions. Specify an integer that indicates the number of seconds to wait. The debugger waits forever if you do not specify a timeout. See <a href="#">"The wait command" on page 207</a> for more information.
s	stack [--context contextid]   [--frame frameid]	Displays current stack information for all contexts. The output includes the frame ID associated with each variable. To display stack information for only a particular context, specify the <code>--context</code> argument. To display stack information for only a particular frame, specify the <code>--frame</code> argument.

Abbreviation	Command	Description
i	<pre>inspect --instance monitorinstance   --instance monitorinstance --frame frameid   --instance monitorinstance --variable variablename   --instance monitorinstance --frame frameid --variable variablename   --frame frameid   --frame frameid --variable variablename</pre>	<p>Displays the value of one or more variables. Specify a monitor instance and/or a frame ID and/or a variable name to display a list of variables in that monitor or in a particular monitor frame, or to display the value of a particular variable. Obtain monitor instance IDs from <code>engine_inspect</code> output or correlator log statements. Obtain frame IDs from <code>engine_inspect stack</code> output.</p>
c	<pre>context [--context contextid]</pre>	<p>Displays information about all contexts in the correlator or about only the context you specify. Information displayed includes context name, context ID, monitor instances in the context, and monitor instance IDs.</p>
e	<pre>enable</pre>	<p>Enables debugging. You must run this in order to do any debugging.</p>
d	<pre>disable</pre>	<p>Disables debugging. You must run this to disable debugging. If you do not disable debugging, the correlator runs more slowly and continues to stop when it hits breakpoints.</p>

Abbreviation	Command	Description
boe	<code>breakonerror enable</code>	Causes the debugger to pause if it encounters an error.
boeoff	<code>breakonerror disable</code>	Causes the debugger to continue processing if it encounters an error.
ba	<pre>breakpoint add [--breakonce] --file filename --line linenumber   [--breakonce] --owner ownername --action actionname --line linenumber</pre>	<p>Adds a breakpoint at the beginning of the specified line. If you do not specify <code>--breakonce</code>, the correlator always pauses at this point when debugging is enabled. You must specify the line number where you want the breakpoint. As usual, this is the absolute offset from the beginning of the file. You must specify either the name of the file that contains the breakpoint or the owner and action name that contains the breakpoint. When the owner is a monitor instance, specify <i>package_name.monitor_name</i> or just <i>monitor_name</i> if there is no package.</p>
bd	<pre>breakpoint delete --file filename --line linenumber   --owner ownername --action actionname --line linenumber   --breakpoint breakpointid</pre>	<p>Removes a breakpoint. Specify one of the following:</p> <ul style="list-style-type: none"> <li>■ File name and line number.</li> <li>■ Owner name, action name and line number. When the owner is a monitor instance, specify <i>package_name.monitor_name</i> or just <i>monitor_name</i> if there is no package.</li> </ul>

Abbreviation	Command	Description
		<ul style="list-style-type: none"> <li>■ Breakpoint ID. You can obtain a breakpoint ID by executing the <code>breakpoint list</code> command.</li> </ul>
bls	<code>breakpoint list</code>	<p>For each breakpoint in the correlator, this displays the following:</p> <ul style="list-style-type: none"> <li>■ Breakpoint ID.</li> <li>■ Name of file that contains the breakpoint.</li> <li>■ Name of the action that contains the breakpoint.</li> <li>■ Name of the owner of the breakpoint.</li> <li>■ Number of the line that the breakpoint is on.</li> </ul> <p>The breakpoint owner is the name of the monitor that contains the breakpoint or the name of the event type definition that contains the breakpoint. If the breakpoint is in an event type definition, the definition must specify an action and processing must create a closure between an event instance and an action call.</p> <p>For information about closures, see "Using action type variables" in <i>Developing Apama Applications</i>.</p>

### Exit status

The `engine_debug` tool returns the following exit values:

Value	Description
0	Success. All requests were processed successfully.
1	Failure. The correlator could not parse the command line, or an exception occurred, such as losing a connection or trying to use a non-existent ID.

## Obtaining online help for the command-line debugger

The command-line debugger provides online help. To obtain general information, enter the following:

```
engine_debug help
```

To get help for a particular command, specify that command after the `help` keyword.

For example, if you want to find out what the `status` command does, enter the following:

```
engine_debug help status
```

Or to find out which options you can specify with the `breakpoint add` command, enter the following:

```
engine_debug help breakpoint add
```

## Enabling and disabling debugging in the correlator

To use the debugger, you must enable debugging in the correlator. To enable debugging locally on the default port, enter the following:

```
engine_debug enable
```

When you are done debugging, you should disable debugging in the correlator. If you do not, the correlator runs more slowly and continues to pause when it hits a breakpoint. To disable debugging in the local correlator on the default port, enter the following:

```
engine_debug disable
```

You can also enable and disable the debugger in a remote correlator by specifying the host name and the port number. For example:

```
engine_debug enable --host foo.bar.com --port 1234
engine_debug disable --host foo.bar.com --port 1234
```

## Working with breakpoints using the command-line debugger

You can use the command-line debugger to add, list and remove breakpoints.

## Adding breakpoints

There are two ways to add a breakpoint. If you know the EPL file name and the line number, you can enter something like the following:

```
engine_debug breakpoint add --file filename.mon --line 27
```

When you specify a file name, you must specify the exact path you specified when you injected the monitor. For example, suppose you ran the following:

```
engine_inject foo.mon
```

You can then specify `foo.mon` for the file name. Now suppose you ran this:

```
engine_inject c:\foo\bar\baz.mon
```

You must then specify `c:\foo\bar\baz.mon` for the file name.

If you prefer to use the monitor and action name, along with the line number, enter something like this:

```
engine_debug breakpoint add --monitor package.monitor --action actionName
--line 27
```

The debugger output indicates the line number where it added the breakpoint. In some cases, the debugger does not set the breakpoint on the line you specified, for example, when a statement runs over multiple lines.

## Listing breakpoints

To obtain a list of the breakpoints currently set in the correlator, enter the following:

```
engine_debug breakpoint list
```

## Removing breakpoints

To remove a breakpoint by specifying the file name and the line number, enter something like the following:

```
engine_debug breakpoint delete --file filename.mon --line 27
```

To use the monitor name to remove a breakpoint, enter something like this:

```
engine_debug breakpoint delete --monitor package.monitor --action actionName
--line 27
```

To delete a breakpoint by using the breakpoint ID that appears in the breakpoint list returned by the debugger, enter something like this:

```
engine_debug breakpoint delete --breakpoint 1
```

## Controlling execution with the command-line debugger

When the correlator stops at a breakpoint, you can use the debugger to step over the next line:

```
engine_debug stepover
```

However, you most likely want to step over the line, confirm that the correlator stopped, and learn about the current state of the debugger. You can do this by entering multiple commands in one line. For example:

```
engine_debug stepover wait --timeout 10 status
```

This is the equivalent of the following three commands:

- `engine_debug stepover` — Causes the debugger to step over one line of EPL.
- `engine_debug wait --timeout 10` — Causes the debugger to pause until either a breakpoint is hit, or ten seconds pass.
- `engine_debug status` — Displays the debugger's current status.

Following are more examples of entering multiple commands in one line.

```
engine_debug stepinto wait --timeout 10 status
engine_debug stepout wait --timeout 10 status
```

To instruct the correlator to continue executing EPL code, run the following command:

```
engine_debug run
```

You use the `engine_debug run` command regardless of how the correlator was stopped — a breakpoint was reached, a step operation, a `wait` command.

To stop the correlator, enter the following command:

```
engine_debug stop
```

## The wait command

The `wait` command connects to the correlator to determine if the correlator has suspended processing. If the correlator is in suspend mode, the `wait` command returns immediately and debugging continues. If the correlator is not in suspend mode, the `wait` command remains connected to the correlator. The `wait` command returns when something else suspends the correlator or when the timeout is reached. Operations that can suspend the correlator include reaching a breakpoint, stepping into or over a line, or some other client explicitly stopping the correlator. If the `wait` command reaches the timeout, it suspends the correlator before it returns.

Stepping can take a variable amount of time. For example, suppose the debugger stops at the end of a listener and you execute a step command. The debugger is now outside the flow of execution until another event comes in. The time that the debugger has to wait for the step to finish is dependent upon when the next matching event arrives.

## Command shortcuts for the command-line debugger

Putting multiple commands in the same command line can get verbose. For example, suppose you want to step out of an action on a remote machine. You would need to enter something like this:

```
engine_debug stepout --host foo.bar.com --port 1234 wait --timeout 10
--host foo.bar.com --port 1234 status --host foo.bar.com --port 1234
```

The command-line debugger provides easier ways to invoke this.

- Any arguments that you specify before the first debugging command apply to the entire command line.
- All individual commands and their arguments have abbreviations.

For example, the following command does the same thing as the previous verbose command:

```
engine_debug -h foo.bar.com -p 1234 sot w -to 10 p
```

The following table lists the abbreviations you can use for command arguments. For abbreviations of commands, see ["Debug commands" on page 200](#).

Command	Abbreviation
--action	-a
--breakonce	-bo
--breakpoint	-bp
--context	-c
--file	-f
--frame	-fm
--host	-n
--instance	-mt
--line	-l
--owner	-o
--port	-p
--raw	-R
--timeout	-to
--utf8	-u
--variable	-v



Command	Abbreviation
<code>--verbose</code>	<code>-V</code>

## Examining the stack with the command-line debugger

When the correlator stops at a breakpoint, you can display the stack with the following command:

```
engine_debug stack
```

The results of this command show the number of the frame that contains each variable. In the following example, the frame number is the number before the right parenthesis:

```
0 )      C:/dev/adbc/apama-test/system/correlator-debug/testcases/
        correctness/Corr_Debug_cor_002/Input/test.mon:35
        foo.baz.test.runtest[master(2)/foo.baz.test(3)]
1 )      C:/dev/adbc/apama-test/system/correlator-debug/testcases/
        correctness/Corr_Debug_cor_002/Input/test.mon:19
        foo.baz.test.:listenerAction::[master(2)/foo.baz.test(3)]
```

You can use these frame numbers (frame IDs) as arguments to the `engine_debug inspect` command.

To see just the contents of the top frame, run this command:

```
engine_debug stack --frame 0
```

## Displaying variables with the command-line debugger

To list all variables in the current stack frame, enter the following:

```
engine_debug inspect
```

To obtain the value for a variable in the current stack frame, enter the following:

```
engine_debug inspect -variable variableName
```

To obtain the value for a variable further down the stack, run the `stack` command to determine the frame number and then enter the following:

```
engine_debug inspect -variable variableName -frame frameid
```

## Generating code coverage information about EPL files

The correlator can generate "code coverage" information about EPL files indicating which lines have been executed. This is useful for measuring the quality of test cases, discovering lines of EPL code which are not being exercised by any tests, as well as for helping diagnose bugs or understand complex interactions in the EPL.

## Recording code coverage information

The recording of code coverage information can be enabled and written (dumped) to disk using management requests, or using an environment variable that automatically writes out a coverage file when the correlator is shut down or when code is deleted from the correlator.

The `epl_coverage` tool can then be used to merge together the coverage files that have been produced by the correlator and produce summary statistics about how much of each source file is covered, as well as an HTML report where each source line is shown annotated with different colors to indicate which lines are not being covered. For detailed information, see ["Creating code coverage reports" on page 212](#).

Enabling the code coverage feature will disable the compiled runtime, and it will also enable the debugger (["Using the command-line debugger " on page 199](#)) and CPU profiler (see ["Using the CPU profiler" on page 189](#)).

### Dumping code coverage information using management requests

One way to enable and dump code coverage information is via the `-r codeCoverage` option of the `engine_management` tool (see also ["Shutting down and managing components" on page 172](#)). You can send the following requests:

Request	Description
<code>codeCoverage on</code>	<p>Enables the recording of code coverage information. This also disables optimizations for any subsequently injected files, disables use of the compiled runtime and enables the EPL debugger. Code coverage must be enabled before injecting EPL to record code coverage information. EPL injected before code coverage is enabled will be omitted from the coverage report (unless using the environment variable as described below).</p> <p><b>Note:</b> This option is not suitable for production use.</p>
<code>codeCoverage off</code>	<p>Disables the recording of code coverage information. This also removes any in-memory coverage information stored so far, but does not reset any features changed by <code>codeCoverage on</code> such as optimizations and possibly the compiled runtime.</p>
<code>codeCoverage dump [filename]</code>	<p>Returns the code coverage information either for all EPL files in the correlator or just for the (optional) source EPL filename provided. The output format is</p>

Request	Description
	suitable for input to the <code>ep1_coverage</code> tool, and is encoded as a UTF-8 string.

### Automatically writing code coverage information using an environment variable

It is also possible to start the correlator in a mode where it automatically writes code coverage information to disk when it is shut down or is given an `engine_delete --all` request (see also ["Deleting code from a correlator" on page 153](#)).

This mode is enabled by setting the `AP_EPL_COVERAGE_FILE` environment variable to the path of a file to which coverage information is to be written. If you do this, the correlator starts in code coverage collection mode with debugging enabled, the compiled runtime disabled and optimizations disabled. On shutdown, it writes the code coverage information to the path specified in the environment variable.

The environment variable can contain replacement tokens in the same format as the correlator log file (see ["Specifying log filenames" on page 139](#)). Given that the coverage file is not subject to log rotation, only the `${PID}` and `${START_TIME}` tags are appropriate.

Example (for Windows):

```
set AP_EPL_COVERAGE_FILE=c:\mypath\mycorrelator.${PID}.ep1coverage
start correlator
(run application, etc.)
engine_management --shutdown "Clean correlator shutdown from command line"
```

Of course, the correlator must be cleanly shut down for this to work, as no coverage information is written if the process is terminated without warning. If a dump is triggered by `engine_delete --all` and more EPL is then injected before the correlator is shut down, all coverage information written by `engine_delete` is overwritten by later coverage information and is thus lost. However, if `engine_delete` is immediately followed by a clean shutdown, there will be no new coverage information when the shutdown occurs. Therefore, the file will not be overwritten.

### Code coverage and deletion of monitors

If you delete an event or monitor, or if the monitor has died or was transient, then this will remove all coverage information associated with that monitor or event, and nothing will be returned by a `codeCoverage dump` request or written to disk automatically on shutdown. You must dump coverage information before the associated code is deleted from the correlator (except when using `engine_delete --all` with the `AP_EPL_COVERAGE_FILE` environment variable, which is a special case that triggers an automatic dump to prevent the information being lost).

### Common usage patterns

- Enable code coverage, inject your application and send typical events into the correlator. Then dump a coverage report. This gives you a complete list of code covered by initialization and events being processed in the system.

- Set the `AP_EPL_COVERAGE_FILE` environment variable before running your test suite. Then collate all the coverage reports. This lets you check that your tests exercise all the code paths.
- Enable code coverage and inject your application. Then disable and enable code coverage (to clear the reporting data). Then send a single event through and dump a coverage report. This lets you see what code is run by a single event.

## Creating code coverage reports

The `epl_coverage` tool takes one or more coverage files that have been output by the correlator's code coverage feature, merges them together to create a new combined `.eplcoverage` file (which can be used as input for the tool), and creates a `.csv` and HTML report of the coverage of each source EPL file. The executable for this tool is located in the `bin` directory of the Apama installation.

### Synopsis

To create code coverage reports, run the following command:

```
epl_coverage [ options ] file1.eplcoverage [ file2.eplcoverage ... ]
```

Example (for Windows):

```
epl_coverage --output c:\mycoverage --source "%APAMA_WORK%\projects\myproject"
--exclude "**/Apama/**/*.*.mon" *.eplcoverage
```

When you run this command with the `-h` option, the usage message for this command is shown.

### Description

The `--output` argument specifies the directory into which the tool writes the output files. If not specified, the current directory is used.

The output includes the following files:

- **merged.eplcoverage.** A single file containing the combined EPL code coverage information from all the input files. This can be used as input to another invocation of the `epl_coverage` tool.
- **coverage\_summary.csv.** Provides a summary of the percentage of lines and instructions covered in each source file in the standard "comma-separated values" text format (in the operating system's local character encoding). This file may be useful for reviewing coverage information in a spreadsheet, or as input for an automated tool that records coverage information as part of a continuous integration build/test system. The file starts with a header line beginning with the hash (#) character which identifies the columns used in the rest of the file. It is recommended that any tool that reads this file should use the header line to identify the contents of each column; this is helpful in case columns are added or reordered in a later release.

- **index.html** (and associated .css and .html files). An HTML summary of coverage information, including annotated copies of the source files showing which executable lines are covered.

The HTML report needs to be able to locate the original EPL source files in order to show an annotated view of them. In most cases, the absolute path of each file is provided when the source file is injected and these files will be found automatically by the tool, provided they were not deleted since the injection. In some cases, however, the full path will not be available, for example, if a source file was injected as part of a CDP (correlator deployment package) file. In such cases, you should use the `--source` option to specify the directory containing the source files. It will recursively search that directory for file names that match the ones which were injected. You should avoid having multiple files with the same names in different directories, else the `--source` searching may find the wrong file.

You can apply filters that remove information about unwanted EPL files from all of the output files (including `merged.epcoverage`). The `--include` and `--exclude` options can each be specified multiple times. They specify file patterns to include or exclude (for example `**/foo/Bar*.mon`). These patterns use the following characters:

- forward slashes (/) to indicate directory separators (on all platforms),
- a single asterisk (\*) to indicate any number of non-directory separator characters,
- two asterisks (\*\*) to indicate any number of characters potentially including directory separators, and
- a question mark (?) to indicate a single character.

If no `--include` argument is provided, the default is to include all file paths, except those that are removed by `--exclude` arguments. These patterns are matched against the absolute paths of the files that were injected into the Correlator, and are not affected by the `--source` argument.

When the number of coverage input files is large, you can avoid an extremely long command line (which some operating systems do not support) by putting the coverage file list into a newline-delimited UTF-8 text file and providing the path to that file on the command line instead, prefixed with an `@` symbol. For example:

```
epl_coverage "@c:\mypath\coverage_file_list.txt"
```

## Options

The `epl_coverage` tool takes the following options:

Option	Description
<code>-h</code>   <code>--help</code>	Displays usage information.
<code>-V</code>   <code>--version</code>	Displays version information for the <code>epl_coverage</code> tool.

Option	Description
<code>-o dir   --output dir</code>	Specifies the directory into which the tool writes the output files. If not specified, the current directory is used.
<code>-i pattern   --include pattern</code>	Filtering option which specifies the EPL source files to include (defaults to <code>**</code> ).
<code>-x pattern   --exclude pattern</code>	Filtering option which specifies the EPL source files to exclude (for example, <code>**/foo/Bar*.mon</code> ).
<code>-s dir   --source dir</code>	HTML report option which specifies the search directory for locating any source files that were injected without specifying an absolute path.
<code>--title str</code>	HTML report option which specifies the title to write into the HTML file.

## Interpreting the HTML code coverage reports

Many lines in an EPL file do not contain any executable instructions, for example, comments, event definitions (except where they contain actions) and event expressions used to declare listeners. These lines are not marked up by the `epl_coverage` tool.

Lines that do contain executable code may have one or more executable elements (instructions), and the `epl_coverage` tool reports whether all or only some of those instructions have been executed. It may therefore be useful to split complex EPL constructs (such as multi-part `if` statements) over multiple lines as much as possible to make the output clearer as to what is covered. The exact details of how many instructions are on any given line is subject to change and therefore not documented, but information on partial coverage may sometimes be useful for identifying branching constructs where not all branches are covered.

**Tip:** Every executable line that is not fully covered has the `(!)` string in the margin, which makes it possible to jump backwards and forwards between these lines using the Find functionality provided by most web browsers.

The purpose of the coverage information is to provide insight into areas of user EPL that are being missed by test cases. Although it is worth aiming for a high percentage of lines and instructions being covered, it is not always possible to write tests that cover every line. However, as long as someone looks at the lines that were missed, there is no need to worry about having less than 100 percent coverage.

Similarly, the information about partial line coverage can often be useful, particularly for control constructs where it might indicate a missed branch in an `if` statement, or

a `while` loop condition that always returns false. But it will not always be possible for users to get 100 percent coverage of every line, or (since the internal instructions used by EPL are not documented and may be changed at any time between versions) even to understand the reason why a line was not fully covered in some cases. Software AG support cannot provide explanations for why a given line of EPL was only partly covered.

## Examples

The following code snippets illustrate some common cases.

- The following line is partially but not fully covered if `a()` returns `true` every time this line is executed, since the instructions for the value of `b()` are never checked in this case.

```
if a() or b() then {
```

- The following line is only partially covered unless the test is run with `DEBUG` logging enabled, since expressions in log statements are only evaluated if the log level is specified.

```
log "Hello world" at DEBUG;
```

- Another common example is a stream query that uses an aggregate where nothing drops out of the window while the test is executed. For example, if less than 100 seconds pass after the first `E()` event, the following line is only partially covered:

```
from a in all E() within 100.0 select com.apama.aggregates.sum(a.val): i
```

If the test does not have anything drop out of the `within` window, then you will get amber coverage, as no code to remove a value from the set being aggregated over (by `sum`) is being executed. This may happen if no events go through this query, or if only less than 20 seconds pass since the first event.

- Any code in an `onunload` action will never be covered at all, since it is only executed with `engine_delete`, which also removes the coverage information.

## Using EPL code coverage with PySys tests

The Apama installation includes the Python-based PySys test framework and some extension modules for Apama components.

The Apama extensions to the PySys test framework can enable code coverage recording, and automatically run a coverage report from the `.eplcoverage` files at the end of test execution, which will help users to create better test cases and to find code paths in their EPL applications that do not have adequate test coverage. To use this feature, start your tests with `-X eplcoverage=true`, for example:

```
pysys run -X eplcoverage=true
```

For an example, see `README.txt` in `software_ag_install_dir /Apama/samples/pysys`.

When recording coverage information, it is important to run PySys without the `--purge` argument. Otherwise, the `.eplcoverage` files will be automatically removed.



## Replaying an input log to diagnose problems

When you start the correlator, you can specify that you want it to copy all incoming messages to a special file, called an input log. An input log is useful if there is a problem with either the correlator process or an application running on the correlator. If there is a problem, you can reproduce correlator behavior by replaying the messages captured in the input log. Incoming messages include the following:

- Events
- EPL
- Java
- Correlator deployment packages (CDPs)
- Connection, deletion, and disconnection requests

If you are unable to diagnose the problem, you can provide the input log to Software AG Global Support. A support engineer can then feed your input log into a new correlator to try to diagnose the problem.

The information in the following topics describes how to generate and use an input log. See also: ["Examples for specifying log filenames" on page 141](#).

## Creating an input log file

To create an input log, specify the following option when you start a correlator:

```
--inputLog filename [${START_TIME}] [${ROTATION_TIME}] [${ID}] [${PID}].log
```

You specify the name of an input log file in the same way that you specify the name of a correlator status log file. See ["Specifying log filenames" on page 139](#).

In addition, specify any other options that you would normally specify when you start the correlator.

## Rotating an input log file

While the input log can get rather large, most file systems can handle large input logs with no special action on your part. However, you might encounter one of the following situations:

- You want to archive your input logs.
- Your operating system enforces a limit on file size.
- The input log has become too large.

In these situations, you can rotate the input log. Rotating the input log means that the correlator closes the current input log and starts sending messages to a new input log.



You should rotate the input log only when you have a specific need to do so. You do not want to have thousands of input logs in a directory since file systems do not handle this efficiently.

If you plan to rotate input logs, specify the `${ID}` tag when you specify the `--inputLog` option when you start the correlator. For examples, see ["Examples for specifying log filenames" on page 141](#).

To rotate the input log, invoke the `engine_management` tool and specify the `--rotateLogs` option. The name of the new input log is the same as the name of the closed input log except that the correlator increments the ID portion of the input log filename by 1. See ["Rotating all log files" on page 198](#).

## Performance when generating an input log

When the file system that hosts the input log is fast, generating an input log should not have any noticeable effect on correlator performance in most cases. It is possible to use the input log with connectivity plug-ins (see *"Working with Connectivity Plug-ins" in Connecting Apama Applications to External Components*), but the performance impact will be significant for chains using the `apama.eventMap` host plug-in and any chains that are using small batches of events. Consequently, the recommendation is to always run correlators that send information to input logs. Just make sure you have enough disk space for the input log. You need to monitor repeated use to determine how much space is required.

With the correlator generating an input log, you can implement your application so that it sends a minimum amount of information to the correlator status log. You do not need to log application information because you can always recover application information from the input log. Implementing an application that sends large amounts of application information to the correlator status log can negatively impact performance.

## Reproducing correlator behavior from an input log

To use an input log to reproduce correlator behavior, you must do the following:

1. Run the `extract_replay_log` Python utility.
2. Run the `replay_execute` script that the `extract_replay_log` utility generates.

### Invoking the extract script

The `extract_replay_log.py` script is in the `utilities` directory in your Apama installation directory. You must have at least Python 2.4 to run this utility. You can download Python from <http://www.python.org>. If you are using Linux, you probably already have Python installed.

The format for running the `extract_replay_log` utility is as follows:

```
extract_replay_log.py [options] inputLogFile
```

Replace *inputLogFile* with the path for the input log you want to extract. If you specify the first input log in a series, the subsequent input logs must be in the same directory as the first input log.

The options you can specify are as follows:

Option	Description
<code>-o=dir   --output=dir</code>	Specifies the directory that you want to contain the output from the <code>extract_replay_log</code> utility. The default is the current directory.
<code>-l=lang   --lang=lang</code>	Specifies the language of the script that the <code>extract_replay_log</code> utility generates. Replace <i>lang</i> with one of the following: <ul style="list-style-type: none"> <li>■ <code>shell</code> to generate the <code>replay_execute.sh</code> UNIX shell script.</li> <li>■ <code>batch</code> to generate the <code>replay_execute.bat</code> Windows batch file. This is the default.</li> </ul>
<code>-c   --correlator</code>	Specifies that the script that <code>extract_replay_log</code> generates should include the command line for starting a correlator. When you run the generated script, the correlator will be started with all of the command line options needed to replay the input log.
<code>--licence</code>	Specifies a path to a licence file for starting a correlator.
<code>--port</code>	Specifies a port on which to start the correlator.
<code>-v   --verbose</code>	Indicates that you want verbose utility output.
<code>-h   --help</code>	Displays help for the utility.

The `extract_replay_log` utility generates the following:

- A script whose execution duplicates the correlator activity captured by the input log.
- Event files where each one is prefixed with `replay_`.
- EPL and possibly JAR and correlator deployment package (CDP) files where each one is prefixed with `replay_`.

### Invoking the replay script

Before you run the replay script, you can optionally edit the generated event files, EPL files, or JAR files to slightly modify the behavior you are about to replay. For example, you might add logging for debugging purposes. However, there are restrictions on what you can change:

- You cannot insert any of the following:
  - calls to `integer.getUnique()` or `rand()`
  - `send`, `emit`, `spawn...to`, `enqueue`, or `enqueue...to` statements
  - `context` constructors
- You cannot change the number of parseable events sent to the correlator. For example, you cannot attach a dashboard component to the input log because the dashboard components work by sending events to the correlator.
- You cannot change the number of event definitions and monitors injected.

Making any of these changes can potentially alter the behavior of later operations.

If you are using the `MemoryStore` and the correlator reads or writes to a store on disk then to accurately play back execution you must have a copy of that store as it was before the correlator modified it. Also, if you are using the `MemoryStore` from multiple contexts it is unlikely to replay correctly because the order of interaction with the `MemoryStore` is not in the input log.

After you have optionally edited the generated files, you are ready to invoke the `replay_execute` script. The `replay_execute` script tries to replay the contents of the input log into the correlator running on the default port.

While the correlator exactly reproduces the activity captured in the input log, it can execute the same activity faster during replay than when it was executed originally. This is because the correlator already has all the events it needs to process; it does not have to wait for any events. Replaying a log is typically significantly faster than original correlator activity. It is possible that you will find that the time it takes to replay a log is not much less than the time it took for the original activity. In this case, it is possible you were running too close to capacity during the original run. If that is the case, you risk not being able to keep up with the event flow during regular correlator execution. If you anticipate higher event flow then you should investigate optimizing your application or running it on a faster computer.

## Event file format

---

You can use the `engine_send` tool to stream a sequence of events through the event correlator. The `engine_send` tool accepts input from one or more data files to support tests or simulations, or from `stdin` to allow dynamic generation of events. In the latter case, you can generate events from user input or by piping output from an event generation program to `engine_send`. In all cases, `engine_send` requires event data

formatted as described in this section. For detailed information on the `engine_send` tool, see ["Sending events to correlators" on page 158](#).

The `engine_receive` tool outputs events in this same file format. This means you can use events generated by the `engine_receive` tool as input to a second event correlator that is executing the `engine_send` tool. For detailed information on the `engine_receive` tool, see ["Receiving events from correlators" on page 162](#).

## Event representation

A single event is identified by the event type name and the values of all fields as defined by that type. Event type names must be fully-qualified by prefixing the package name into which the corresponding event type was injected, unless the event was injected into the default package.

Each event is given on a separate line, separated by a new-line character. Only single-line comments are allowed. Start each comment line with `//` or `#`. Any blank lines are ignored.

For example, following are three valid events:

```
// This is an event file that contains some sample events.
// Here are three stock price events:
StockPrice("XRX", 11.1)
StockPrice("IBM", 130.6)
StockPrice("MSFT", 70.5)
```

For those events, the following event type definition must be injected into the default package:

```
event StockPrice {
    string stockSymbol;
    float stockValue;
}
```

If the above events were saved in an `.evt` file, `engine_send` would send each event in turn, as soon as the previous event finished transmission. This behavior can optionally be modified in several ways:

- Specifying that batches of events should be sent at specified time intervals.
- Specifying that all events on all queues should be processed before sending the next event.

## Event timing

In `.evt` files, it is possible to specify the following:

- Time intervals for sending batches of events to the correlator.
- Waiting for all events on all queues at that point in time to be processed before sending the next event.

### Adding `BATCH` tags to send events at intervals

You can specify time intervals for sending batches of events to the correlator. This is achieved by specifying the `BATCH` tag followed by a time offset in milliseconds. For example, the following specifies two batches of events to be sent 50 milliseconds apart.

```
BATCH 50
StockPrice("XRX", 11.1)
StockPrice("IBM", 130.6)
StockPrice("MSFT", 70.5)
BATCH 100
StockPrice("XRX", 11.0)
StockPrice("IBM", 130.8)
StockPrice("MSFT", 70.1)
```

The addition of a "time" allows simulations of "bursts" of events, or more random distributions of event traffic. Times are measured as an offset from when the current file was opened. If only one file of events is being read and transferred, then this would be the same as since the start of a run (that is, from the time that the `engine_send` tool starts processing the event data). If multiple files are being read in, the timing starts all over again upon the (re)opening of each file.

If the time given for a batch is less than the current time, or if no time is given following a `BATCH` tag (or if no `BATCH` tag is provided), then the events are sent as soon as they are read in, immediately following the preceding batch.

### Using `&FLUSHING` mode for more predictable event processing order

Sending events in flushing mode can help provide a more predictable event processing order. However, flushing mode is slower than the default behavior.

By default, events are delivered in an optimal way, not waiting for previously sent events to be processed before the next event is delivered to contexts (or other consumers of channels). When flushing mode is enabled the behavior is as follows:

1. The correlator sends an event.
2. The correlator processes all events on all queues at that point in time, repeating this as many times as specified in the flushing specification.
3. The correlator sends the next event.

To enable flushing mode, insert the following line in a `.evt` file:

```
&FLUSHING ( n )
```

Replace `n` with an integer that specifies how many times to flush queues in between each event. Set this to the maximum length of a chain of send-to operations between contexts that could occur in your application. If you specify a number that is bigger than required the correlator simply repeats the flush operation, which incurs a small overhead. To disable flushing mode, insert the following line in the `.evt` file:

```
&FLUSHING ( 0 )
```

Enabling or disabling flushing mode affects only the events sent on that connection or from that event file.

When sending `&TIME` events in to a multi-context application, the time ticks are delivered directly to all contexts. This can be different than the way in which events in the `.evt` file are sent in to the correlator and then sent between contexts in an application. This difference can result in processing events at an incorrect simulated time. In these cases, sending `&FLUSHING(1)`, for example, before sending time ticks and events can result in more predictable and reliable behavior.

## Event types

The following example illustrates how each type is specified in an event representation. Given the event type definitions:

```
event Nested {
    integer i;
}
event EveryType {
    boolean b;
    integer i;
    float f;
    string s;
    location l;
    sequence<integer> si;
    dictionary<integer, string> dis;
    Nested n;
}
```

the following is a valid event representation for an `EveryType` event:

```
EveryType (
    true,           # boolean is true/false (lower-case)
    -10,            # positive or negative integer
    1.73,           # float
    "foo",          # strings are (double) quoted
    (1.0,1.0,5.0,5.0), # locations are 4-tuples of float values
    [1,2,3],        # sequences are enclosed in brackets []
    {1:"a",2:"b"},  # dictionaries are enclosed in braces {}
    Nested(1)       # nested events include event type name
)
```

Note that this example is split over several lines for clarity; in practice this definition would all be written on the same line.

Types can of course be nested to create more complex structures. For example, the following is a valid event field definition:

```
sequence<dictionary<integer, Nested> >
```

and the following is a valid representation of a value for this field:

```
[{1:Nested(1)}, {2:Nested(2)}, {3:Nested(3)}]
```

## Event association with a channel

The `engine_send` tool can send an event file that associates channels with events. Likewise, the `engine_receive` tool can output an event file that includes the channel on which an event was received. The event format is the same for both tools:

```
"channel_name",event_type_name(field_value1[, field_valuen]...)
```

For example, suppose you want to send `Tick` events, which contain a `string` followed by an `integer`, to the `PreProcessing` channel. The contents of the `.evt` file would look like this:

```
"PreProcessing",Tick("SOW", 35)
"PreProcessing",Tick("IBM", 135)
```

A channel name is optional. In a file being sent with the `engine_send` tool, you can mix event representations that specify channels with event representations that do not specify channels. Events for which a channel is specified go to only those contexts subscribed to that channel.

The default behavior is that events are sent on the default channel (the empty string) when a channel is not explicitly specified. Events sent on the default channel go to all public contexts. All running Apama queries receive events sent on the default channel. To change the default behavior for events sent by the `engine_send` tool, you can specify `engine_send -c channel`. If a channel is not explicitly specified for an event, then it is sent to the channel identified with the `-c` option. See ["Sending events to correlators" on page 158](#).

## Using the data player command-line interface

Apama's Data Player in Software AG Designer lets you 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. Software AG Designer back event data that has been stored in standard data formats.

When you are ready to test your application, the command-line interface to the Data Player lets you write scripts and unit tests to exercise the API layers. Or, if you just want to play back events to the correlator, using the command-line interface might be easier than using the Data Player GUI in Software AG Designer.

To use the command-line interface to the Data Player, you must have already used the GUI interface in Software AG Designer. That is, you must have already defined queries and query configurations in Software AG Designer. When you use the command-line interface (that is, the `adbc_management` tool), you specify query names and query configurations that you created in Software AG Designer. The executable for this tool is located in the `bin` directory of the Apama installation.

The Data Player relies on Apama Database Connector (ADBC) adapters that are specific to standard ODBC and JDBC database formats as well as the comma-delimited Apama Sim format. Apama release 4.1 and earlier captured streaming data to files in the Sim format. These adapters run in the Apama Integration Application Framework (IAF), which connects the data sources to the correlator. The information here assumes that you are already familiar with the information in "Using the Data Player" in *Using Apama with Software AG Designer*.

## Synopsis

To use the Data Player from the command line, run the following command:

```
adbc_management --query queryName --configFile file [ options ]
```

When you run this command with the `-h` option, the usage message for this command is shown.

## Options

The `adbc_management` tool takes the following options:

Option	Description
<code>-h   --help</code>	Displays usage information.
<code>-n host   --hostname host</code>	Name of the host on which the event correlator is running. The default is <code>localhost</code> . Non-ASCII characters are not allowed in host names.
<code>-p port   --port port</code>	Port on which the event correlator is listening. The default is 15903.
<code>--query queryName</code>	Runs the specified query, which is defined in the query configuration file that you identify with the <code>--configFile</code> option. This is a query you created with Apama's Data Player in Software AG Designer. You did this when you clicked on the + button on the action bar. You specified a query name, and that is the name you need to specify here.
<code>--configFile file</code>	The query configuration file to use. This is the query configuration file associated with your project. In Software AG Designer, the query configuration file is always called <code>dataplayer_queries.xml</code> (in the project's <code>config</code> directory).
<code>--username user</code>	The user name to use for the database connection. Optional.
<code>--password password</code>	The password to use for the database connection. Optional.
<code>--returnType returnType</code>	The type of the playback events returned. The default is <code>Native</code> . The only other choice is <code>Wrapped</code> . A return type of <code>Native</code> means that each matching event is sent as-is to the correlator. When you



Option	Description
	specify <code>Wrapped</code> , each matching event is inside a container event. The name of the container event is <code>Historical</code> followed by the name of the event in the container, for example, <code>HistoricalTick</code> . The container event will be in the default namespace. Event wrapping allows events to be sent to the correlator without triggering application listeners. A separate, user-defined monitor can listen for wrapped events, modify the contained event, and reroute it such that application listeners can match on it.
<code>--backTest <i>boolean</i></code>	This option is equivalent to Software AG Designer's Data Player option to <b>Generate time event from data</b> . When the correlator is running with the <code>-Xclock</code> option, time in the correlator is controlled by <code>&amp;TIME()</code> events. This is how the Data Player controls the playback speed. If the correlator is not running with the <code>-Xclock</code> option, the correlator keeps its own time. The default is <code>true</code> , which means that the correlator is running with the <code>-Xclock</code> option. Set this option to <code>false</code> when the correlator is not running with the <code>-Xclock</code> option.
<code>--speed <i>playBackSpeed</i></code>	Specifies the speed for playing back the query. Optional. A float value less than or equal to <code>0.0</code> means that you want the correlator to play it back as fast as possible. A float value greater than <code>0.0</code> indicates a multiple for the playback speed. To play at normal speed, specify <code>1.0</code> . For half normal speed, specify <code>0.5</code> . For twice normal speed, specify <code>2.0</code> . For 100 times normal speed, specify <code>100.00</code> .

## Using the Apama component extended configuration file

The Apama component extended configuration file is an optional file that you can use to do the following:

- Bind Apama server components to a particular set of addresses.
- Specify that Apama client connections must be from a particular IP address or range of IP addresses.
- Set environment variables for Apama components.

- Set log files and log levels for EPL root, packages, monitors, or events.

You can specify the optional extended configuration file when you start the correlator. If you do, the settings in the extended configuration file apply to the following Apama components:

- Correlator
- IAF
- Sentinel Agent
- `engine_receive` correlator tool

In an extended configuration file, if a line includes a special character that you want to be treated as a literal, you must escape it by inserting a backslash just before it. A character that follows two consecutive backslashes (`\\`) is treated as a literal. Single quotes inside double quotes are treated as a literal. Double quotes inside single quotes are treated as a literal. See the example in ["Sample extended configuration file" on page 229](#).

## Binding server components to particular addresses

To bind Apama server components to a particular address or set of addresses, specify a `BindAddress` line for each address. Specify this in the `[Server]` section of the extended configuration file. For example:

```
[Server]
BindAddress=127.0.0.1:15903
BindAddress=10.0.0.1
```

You can specify as many `BindAddress` lines as you want. Clients can connect to any of the listed addresses.

An IP address is required. If you do not specify a port, the Apama server components use the port that is specified when the correlator is started. This makes it possible to share extended configuration files if you want to restrict connections according to only addresses.

If you do not specify an extended configuration file when you start the correlator, or there are no `BindAddress` entries in the extended configuration file, the Apama components bind to the wildcard address (`0.0.0.0`).

## Ensuring that client connections are from particular addresses

To ensure that client connections are from particular addresses, add one or more `AllowClient` entries to the extended configuration file in the `[Server]` section. For example:

```
[Server]
AllowClient=127.0.0.1
AllowClient=192.168.128.0/17
```

An `AllowClient` entry takes an IP address, as in the first example above, or a CIDR (Classless Inter-Domain Routing) address range, as in the second example above. With these example entries in the extended configuration file, the Apama components allow connections from either the localhost (127.0.0.1) or IP addresses where the first 17 bits match the first 17 bits of 192.168.128.0. The Apama components do not accept connections from any other IP addresses. This creates a "whitelist" of allowable IP addresses.

If you specify an extended configuration file when you start the correlator, and if there are any `AllowClient` entries in the extended configuration file, then the Apama components do not allow connections from any IP address that does not fall within one of the `AllowClient` ranges specified. If you do not specify an extended configuration file when you start the correlator, or there are no `AllowClient` entries in an extended configuration file that you do specify, the Apama components accept connections from any client.

**Important:** This feature is intended to prevent mistakenly connecting to the wrong server. It is not intended to prevent malicious intruders since it provides no protection against address spoofing.

## Setting environment variables for Apama components

You can use the extended configuration file to set environment variables. Put environment variable declarations in the `[Environment]` section. For example:

```
[Environment]
MY_ENV_VAR=myvalue
```

If you specify an extended configuration file when you start the correlator, and if there are any environment variable entries in the extended configuration file, then the Apama components use those environment variable settings. If you do not specify an extended configuration file when you start the correlator, or there are no environment variable entries in an extended configuration file that you do specify, the Apama components use the environment variable settings specified elsewhere.

**Note:** You cannot use this feature to set variables such as `LD_PRELOAD` and `LD_LIBRARY_PATH` because UNIX dictates that they are set before the affected process starts execution. These environment variables should therefore be considered read-only.

## Setting log files and log levels in an extended configuration file

You can configure per-package logging in two ways:

- Statically, in the extended configuration file when starting the correlator, as described here.

- Dynamically, using the following `engine_management` options: `--setApplicationLogFile` and `--setApplicationLogLevel`. See ["Setting logging attributes for packages, monitors and events" on page 194](#).

To set log files and/or log levels for EPL root, packages, monitors, or events, specify entries in the `[ApplicationLogging]` section of the extended configuration file.

To set the default log file and level for the EPL root package, specify two lines in the following format:

```
RootFile=rootLogFilename
RootLevel=ROOTLOGLEVEL
```

Replace *rootLogFilename* with the name of the log file for the EPL root package. No spaces are allowed in the log file name. Replace *ROOTLOGLEVEL* with `TRACE`, `DEBUG`, `INFO`, `WARN`, `ERROR`, `FATAL`, `CRIT` or `OFF`. For example:

```
RootFile=apama\\root.log
RootLevel=ERROR
```

Note that you must insert a backslash to escape special characters. In the previous example, there is a filepath that contains a backslash that must be escaped. Hence, the double backslash. You do not need to specify both a log file and a log level; you can specify one or the other. If you do not specify a log file or log level for the root package, it defaults to the correlator's log file/log level.

To set the default log file and level for an EPL package, monitor, or event, specify two lines in the following format:

```
PackageFile.node=nodeLogFilename
PackageLevel.node=NODELOGLEVEL
```

Replace *node* with the name of the EPL package, monitor, or event you are setting the logging attribute for. If a monitor or event is in a named package and not the root package, be sure to specify the fully qualified name.

Replace *nodeLogFilename* with the name of the default log file for the specified EPL package, monitor or event. No spaces are allowed in the log file name. Replace *NODELOGLEVEL* with `TRACE`, `DEBUG`, `INFO`, `WARN`, `ERROR`, `FATAL`, `CRIT` or `OFF`. This is the default log level for the specified node.

For example:

```
PackageFile.com.myCompany.Client=apama\\Client.log
PackageLevel.com.myCompany.Client=DEBUG
```

For a particular node, you do not need to specify both a log file and a log level; you can specify one or the other. If you do not specify a log file or log level for a particular node, it defaults to the settings for a parent node. See ["Tree structure of packages, monitors, and events" on page 194](#).

There is no limit to the number of log settings that can appear in the configuration file. The last line that refers to a given node takes priority over earlier lines. When you set log attributes in the extended configuration file, the rules for hierarchical logging apply. See ["Setting logging attributes for packages, monitors and events" on page 194](#).

If you pass an extended configuration file to a correlator when you start that correlator and the configuration file contains an `[ApplicationLogging]` section, the correlator uses the logging settings in that section. If you do not pass a configuration file when you start a correlator, or there are no settings in the `[ApplicationLogging]` section, then correlator initialization does not include any log settings except for the default correlator log.

Whether or not you specify an extended configuration file when you start a correlator, any log settings you specify can be overwritten after initialization by invoking the `engine_management` tool and specifying the `--setApplicationLogFile` and/or `--setApplicationLogLevel` options. See ["Managing application log levels" on page 195](#) and ["Managing application log files" on page 196](#).

## Sample extended configuration file

Save the extended configuration file as a text file. Blank lines are ignored. For example, the contents of `ApamaExtendedConfig.txt` might be as follows:

```
[Server]
BindAddress=127.0.0.1:15903
BindAddress=10.0.0.1
AllowClient=127.0.0.1
AllowClient=10.0.0.0/24
[Environment]
LD_LIBRARY_PATH=/usr/local/mydir
[ApplicationLogging]
PackageFile.com.myCompany=apama\apama.log
PackageLevel.com.myCompany=WARN
PackageLevel.com.myCompany.Client=DEBUG
```

## Starting a correlator with an extended configuration file

To use an Apama component extended configuration file, specify the `-Xconfig` option with the extended configuration file path when you start the correlator. For example, if both the license file and the extended configuration file are in the same directory as the correlator executable, and the name of the extended configuration file is `ApamaExtendedConfig.txt`:

```
run correlator -l ApamaServerLicense.xml -Xconfig ApamaExtendedConfig.txt
```