Release Notes

Version 9.10

August 2016

APAMA

Document ID: PAM-RN-910-20160810

Table of Contents

About this Guide

Release Notes describes the changes introduced with the current Apama release as well as earlier releases.

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
Release Notes	Describes new features and changes since the previous release.
Installing Apama	Instructions for installing Apama.
Introduction to Apama	Introduction to developing Apama applications, discussions of Apama architecture and concepts, and pointers to sources of information outside the documentation set.
Using Apama with Software AG Designer	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.
Developing Apama Applications	Describes the different technologies for developing applications: EPL monitors, Apama queries, Event Modeler, and Java. You can use one or several of

Title	Description
	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.
Connecting Apama Applications to External Components	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: ■ Implement standard Apama Integration Adapter Framework (IAF) adapters. ■ Create applications that use correlator-integrated messaging for JMS or Software AG's Universal Messaging. ■ Use connectivity plug-ins written in Java or C++. ■ Develop adapters with Apama APIs for Java and C ++. ■ Develop client applications with Apama APIs for Java, .NET, and C++.
Building and Using Dashboards	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.
Deploying and Managing Apama Applications	Describes how to deploy components with Command Central or with Apama's Enterprise Management and Monitoring (EMM) console. Also provides information for: ■ Improving Apama application performance by using multiple correlators and saving and reusing a snapshot of a correlator's state. ■ Managing and monitoring over REST (Representational State Transfer).

Title	Description
	■ Using correlator utilities.

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 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 and give us a call.

1 License Terms and Technical Restrictions

Using Apama 9.10 (and higher) without a valid license file means the product will function as Apama Community Edition. This binds the user and any related organization to the Freemium license terms in the Software AG license agreement (refer to the license agreement for further details). In addition, the following Apama Community Edition restrictions also apply:

- Support is not available from Software AG's Empower support portal.
- Projects cannot incorporate more than 10 correlators.
- Attribution is required in the end application's Help/About screen or relevant alternative (for example, "Streaming analytics by Software AG Apama").
- The correlator will execute EPL on a maximum of 4 CPU threads (and so performance may be restricted).
- The correlator does not permit the "distributed" mode of the MemoryStore (that is, in-memory store only).
- The correlator does not permit processing Apama queries against distributed stores such as Software AG BigMemory (that is, in-memory store only).
- Correlator-integrated messaging for JMS only permits best-effort messaging (that is, reliable messaging is not permitted).
- The correlator log file includes notes highlighting it is running without a valid license file.
- The correlator will terminate if its resident memory usage rises above 1024MB.
- The correlator only permits 20 contexts to be created (and so performance may be restricted).
- The correlator only permits 5 EPL monitor types to be persistent.
- The correlator cannot read user-generated correlator deployment packages (CDPs).
- The correlator only handles 5 Apama query definitions.
- The correlator only handles 5 instances of each Apama query definition.
- The correlator only handles 50 unique partitions for an Apama query.

See also "Running Apama without a license file" in *Introduction to Apama*.

If you need to remove these Apama Community Edition restrictions, then please contact Software AG to purchase Apama.

2 What's New In Apama 9.10

Apama 9.10 runs on the platforms listed in the *Supported Platforms* document for version 9.10. This is available from the following web page: http://documentation.softwareag.com/apama/index.htm. Be sure to consult that document for details about supported versions of operating systems, compilers, BigMemory, and Universal Messaging.

Apama 9.10 includes new features, enhancements, and changes as described in the following topics.

New connectivity plug-ins feature in 9.10

Apama now provides a new way for developing adapters called *connectivity plug-ins*.

Connectivity plug-ins can be written in Java or C++, and run inside the correlator process to allow messages to be sent and received to/from external systems. Individual plug-ins are combined together to form *chains* that define the path of a message, with the correlator host process at one end and an external system or library at the other, and with an optional sequence of message mapping transformations between them. A configuration file describes both the chains and the plug-ins making up each chain. The configuration file is written using the YAML markup language, and can express structured configuration (maps, lists and simple values) for plug-ins. For detailed information, see "Using Connectivity Plug-ins" in *Connecting Apama Applications to External Components* .

A new option, `--connectivityConfig`, is available for the `correlator` tool, which specifies the path to the above mentioned configuration file. See "Starting the event correlator" in *Deploying and Managing Apama Applications* .

A new predefined annotation, `com.softwareag.connectivity.ExtraFieldsDict`, is available, which can be used to place map keys that do not name fields into a dictionary. See "Adding predefined annotations" in *Developing Apama Applications*.

New query and DataView metadata feature in 9.10

You can now record information about a query in the metadata section. This can be, for example, the recording author, the version number, or the last modified date of a query. See "Defining metadata in a query" in *Developing Apama Applications* for more detailed information.

Metadata properties can be specified for a DataView by adding keys with the prefix `DataViewDefinition.EXTRA_PARAMS_METADATA_PREFIX` to the `extraParams` dictionary of `DataViewAddDefinition` when adding the new DataView definition.

A new constant, `EXTRA_PARAMS_METADATA_PREFIX`, has been added to the ScenarioService API. Any metadata properties that were set as part of a query or DataView definition will appear in the `extraParams` with this prefix. See the `IScenarioDefinition` interface in the *API Reference for Java (Javadoc)* or *API Reference for .NET* for more information.

Once defined, metadata information about a query can be viewed in the Scenario Browser. The Scenario Browser shows metadata properties for queries and DataViews. Metadata properties are shown on the main panel when the query node is selected.

See also "Miscellaneous enhancements and changes in 9.10" on page 21 for information on irrelevant sections in the Scenario Browser that are now hidden from the main panel.

See also "Dashboard enhancements in 9.10" on page 21 for information on the new "Definition extra params table" which can be used to display metadata (and other `extraParams` information) in a dashboard.

New EPL memory profiler feature in 9.10

An EPL memory profiler is now provided that you can use to display information on monitors and monitor instances. This is helpful for checking the performance and for finding memory leaks.

To make use of the EPL memory profiler, you use the `-r` (or `--dorequest`) option of the `engine_management` tool with one of the following request types:

- `eplMemoryProfileOverview`
- `eplMemoryProfileMonitorInstanceDetail`
- `eplMemoryProfileMonitorDetail`

The information that is returned for a request 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. For detailed information, see "Using the EPL memory profiler" in *Deploying and Managing Apama Applications*.

New EPL code coverage feature in 9.10

The correlator can now 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.

The recording of code coverage information can be enabled and written 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 new `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.

The Apama extensions to the PySys test framework can now enable code coverage recording, and automatically run a coverage report 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.

For detailed information, see "Generating code coverage information about EPL files" in *Deploying and Managing Apama Applications*.

New PySys version in 9.10

Apama 9.10 ships a new release of PySys, version 1.1.0. This new release includes many useful fixes and enhancements. For full details, see the PySys release notes.

One significant addition in PySys 1.1 is an optional `abortOnError` capability that produces clearer and more timely "outcome reason" messages to explain why a testcase has failed when something goes wrong, such as an injection failure or an expected signal not being found in a file. The `abortOnError` functionality can be enabled by setting the following property in your project file:

```
<property name="defaultAbortOnError" value="true"/>
```

See the pysysproject.xml file in the samples/pysys directory of your Apama installation for examples of all the settings we recommend for all new test projects. Existing projects may wish to continue with the older settings to avoid the need to fix up any failing tests.

Support has been added for generating EPL code coverage reports from PySys test executions. See "Using EPL code coverage with PySys tests" in *Deploying and Managing Apama Applications* for more details.

Correlator and IAF components are now shut down cleanly by default at the end of each test execution, which is a prerequisite for getting EPL code coverage information. See also "Miscellaneous changes in 9.10 affecting backwards compatibility" on page 24 for information on how to disable this new behavior.

A new `antBuild` method has been added which can be used to invoke Apache Ant builds with the Apama environment variables defined. This can be used, for example, to execute an exported build.xml file which references the apama-macros.xml file.

Several new methods have been added to the Apama PySys correlator object:

- `inspect`
- `sendEventStrings`
- `inputLog` argument to `startCorrelator`

On Linux, PySys Docker extensions are now available in the third_party/python/lib/python2.7/site-packages/docker directory of your Apama installation.

New dashboard server HTTP REST support in 9.10

You can now monitor Apama dashboard data server and display server components with the same Apama Representational State Transfer (REST) HTTP protocol that is already supported by the correlator and IAF. This provides monitoring capabilities to third-party managing and monitoring tools or to any application that supports sending and receiving XML or JSON documents over the HTTP protocol.

The HTTP protocol uses the management port of the dashboard servers. As a result, in addition to the existing `dashboard_management` tool, it is now possible to use the `component_management` tool, the Java `GenericComponentManagement` API or any HTTP REST client supporting XML or JSON to perform tasks such as pinging the server, listing the version number, memory usage, environment variables and start-up arguments, shutting down the server, and sending other dashboard-specific requests. See "Managing and Monitoring over REST" in *Deploying and Managing Apama Applications* for more information about using REST with Apama.

New JSON-based REST support in 9.10

Support has been added for the JSON equivalent to most XML URI `GET` requests as per those listed below:

- `/correlator/info`
- `/correlator/status`
- `/correlator/appLogging`
- `/correlator/types`
- `/iaf/status`
- `/info`
- `/logLevel`
- `/connections`

For detailed information, see "Managing and Monitoring over REST" in *Deploying and Managing Apama Applications*.

The correlator types are now also accessible from the web browser using http://localhost:15903/correlator/types. Note that there is no security around these requests. Anyone who has access to the host and port will be able to make these requests. See "Security Requirements for Apama" in *Deploying and Managing Apama Applications* for more information.

Command Central support in 9.10

A **Platform Manager Plug-in** node is now available in the product selection tree of the Software AG Installer. You can see it when you expand the **Apama** node. This plug-in allows you to manage the server components of Apama using Command Central.

For more information, see the Command Central documentation at http://documentation.softwareag.com/ (Empower login required). You can find it under *Guides for Tools Shared by Software AG Products*.

Apama integration with Command Central supports the following features:

- **Inventory actions**. Creating and deleting an Apama instance representing the configuration needed to start a correlator, IAF, dashboard data server or display server process.
- **Configuration Manager**. Update instance configuration.
- **Lifecycle operations**. Start, stop, and restart an Apama instance (correlator, IAF, dashboard data server and dashboard display server instances).
- **Monitoring components** Monitoring correlator, IAF, dashboard data server and dashboard display server instances.

For more information, see "About deploying components with Command Central" in *Deploying and Managing Apama Applications*.

Apama enhancements in Software AG Designer 9.10

Apama 9.10 includes the following enhancements in Software AG Designer:

- The times for doing incremental rebuilds of large Apama projects have been significantly reduced. Tests have shown that this has been speeded up by more than a factor of two on a typical desktop.
- When configuring a JDBC adapter, you can now specify the timeout for queries in the **General Properties** section. The new property **Query timeout** is available for this purpose. Keep in mind that different database vendors define a query timeout differently; see the documentation for these databases for more information. See also "Configuring an ADBC adapter" in *Connecting Apama Applications to External Components*.
- You can now enable or disable diagnostic logging for a query file. This feature was available only for scenarios and was labelled as **Generate Debug Code**. In this release, the label **Generate Debug Code** is renamed to **Diagnostic Logging**.
- The top-level page of the Apama preferences now informs you in which directory you can find the Apama log file for Software AG Designer, and a link is now

provided for accessing that directory directly. See also the description of the "Apama" preferences page in *Using Apama with Software AG Designer*.

EPL enhancements in 9.10

Apama 9.10 includes the following EPL enhancements:

- A new method `add()` has been added to the `Table` class of the MemoryStore. `add()` does the same as `get()`, except that it does not check if the row that is to be added already exists in the table until `commit()` is called and it therefore never throws an exception. If you are sure that the row does not yet exist, you can use `add()` as this is faster than `get()`. See also "Creating and removing rows" in the description of the MemoryStore in *Developing Apama Applications*.
- All comparable types can now be used with the comparison operators (`<`, `>`, `<=`, `>=`, `=` and `!=`). This includes sequences and dictionaries where all members are of a comparable type. See also "Comparable types" in *Developing Apama Applications*.
- The following new built-in aggregate functions are now available:
  - `countUnique(`*`value`*`)`
  - `percentile(`*`value`*`)`

  The built-in positional aggregate functions `first(`*`value`*`)`, `last(`*`value`*`)` and `nth(`*`value`*`,` *`index`*`)` now also support `integer`, `string`, `boolean` and `location` types. In addition, `nth(`*`value`*`,` *`index`*`)` now also supports negative indices to get items from the end of the window (`-1` means the last item, `-2` means the second last item, and so on).

  The built-in aggregate functions `min(`*`value`*`)` and `max(`*`value`*`)` now also support `integer` types.

  See also "Built-in aggregate functions" in *Developing Apama Applications*.

Correlator utility enhancements in 9.10

Apama 9.10 includes the following correlator utility enhancements:

- Using the compiled runtime now requires that the binutils package is installed on the Linux system. For the Linux platforms supported by Apama, this is part of the default installation.

  Injection times of applications using the compiled runtime have been improved in this release. There is a performance improvement in initial injections for codebases with a large number of actions.

  The correlator can also be configured to use a cache of the compiled artifacts to speed up subsequent re-injection of the same source files. For this purpose, the following new option is available for the `correlator` tool:

```
--runtime-cache dir
```

For more information, see "Starting the event correlator" and "Injection time of compiled runtime" in *Deploying and Managing Apama Applications*.

- Using the new optional `${PID}` tag, you can now add the process ID to the name of a log file. This is helpful if you want to run multiple correlators with the same arguments but with separate log files. For more information, see "Specifying log filenames" in *Deploying and Managing Apama Applications*.
- New options for log handling which were previously only available as request types of the `-r` option are now available with the `engine_management`, `component_management` and `iaf_management` tools.

  The following options are available for all of the above-mentioned tools:

  ```
  --setLogFile path

  --reopenLog
  ```

  The following options are only available for the `engine_management` tool:

  ```
  --rotateLogs

  --setApplicationLogLevel [package=]level

  --setApplicationLogFile [package=]path

  --unsetApplicationLogLevel package

  --unsetApplicationLogFile package

  --unsetRootApplicationLogLevel

  --unsetRootApplicationLogFile

  --getApplicationLogLevel package

  --getApplicationLogFile package

  --getRootApplicationLogLevel

  --getRootApplicationLogFile
  ```

  For more information on these options, see "Shutting down and managing components" in *Deploying and Managing Apama Applications*.
- The following new options are now available for the `engine_management`, `component_management` and `iaf_management` tools, which can be used to get the uptime, and the virtual and physical memory usage of the component:

  ```
  -M | --getuptime

  -Vm | --getvmemory

  -Pm | --getpmemory
  ```

  For more information on these options, see "Shutting down and managing components" in *Deploying and Managing Apama Applications*.

Corresponding methods are also available in the Java EngineClient API.

- As of 9.10.0.4, it is possible to deploy Apama applications with a YAML configuration file. See "About deploying Apama applications with a YAML configuration file" in *Deploying and Managing Apama Applications*. For this purpose, the new `--config` option is available for the `correlator` tool. It is used to specify the name of the YAML file that lists files to inject at startup. This is limited to .mon, .cdp, .jar and .evt files, and the files must be listed in the correct injection order. This is provided as an alternative mechanism for initializing a correlator for Docker or environments where it is not practical to run Java such as the Apama macros.

Dashboard enhancements in 9.10

Apama 9.10 includes the following dashboard enhancements:

- The Google map object is now available on the **Graphs** tab of the Dashboard Builder's Object Palette. This object is available in Thin Client deployments only.
- The **Table (HTML5)** table object is now available on the **Tables** tab of the Dashboard Builder's Object Palette. This is an advanced HTML implementation of the table object which provides features such as enhanced column filtering, column re-ordering and column locking. This object is available in Thin Client deployments only.
- A new tabular data attachment type "Definition extra params table" is now available. This new table shows the metadata of the `extraParams` member of a `definition` (scenario, DataView or query). An entry in `extraParams` is considered metadata if its `key` name starts with the `Metadata:` prefix. This prefix will be automatically added for the query properties.
- A new command line argument `--dashboardDir` *`folder`* is now available. The Display Server can be run from any directory by using this option to specify an arbitrary *`folder`* where deployed dashboards can be found. If this option is not used, then `APAMA_WORK/dashboards` is assumed.
- A new command line argument `--dashboardExtraJars` *`jarFiles`* is now available. Custom function, custom command and/or any extra jar files referenced by the dashboard can now be specified by using this `--dashboardExtraJars` option. *`jarFiles`* contains a list of semicolon-separated JAR files to be added to the classloader at runtime. If this option is not used, the `APAMA_DASHBOARD_CLASSPATH` environment variable must be set accordingly for the JAR files to be accessible by dashboard processes.

Miscellaneous enhancements and changes in 9.10

- It is now possible to pass the username and password on the JDBC connection URL, rather than having to set it in EPL. To do so, you must set the special value

ADBC_NULL on both the username and password parameter on the EPL side, for example, via the Connection.openDatabaseFull() action in ADBCEvents.mon.

- The new IAFStatusDataViewService provides support for publishing the status of an IAF adapter as a DataView item. It can be used to easily monitor the adapter status using Apama's Scenario Browser or to visualize the adapter status using an Apama dashboard. For more information, see "DataView support" in *Connecting Apama Applications to External Components*.
- ApamaDoc documentation is now provided for most of Apama's public event APIs.
- By default, the ApamaDoc export wizard does not generate documentation for inner fields of a monitor. If you want to include inner fields of a monitor, run ApamaDoc in headless mode using the --includeMonitorMembers option.
- ApamaDoc now supports <code/> tags inside ApamaDoc comments.
- The correlator now logs INFO log lines if an external sender sending events in to the correlator is blocking, giving a reason as to why the queue is full. For example:

```
Blocked adding 23 events to context main (id=1)'s queue; context is blocked
 waiting on correlator (on port 62156) which is flagged as a slow receiver
Blocked adding 2 events to context main (id=1)'s queue; context is blocked
 waiting on ctx1 (id=2) which is running a plug-in
```

- The Java APIs now implement the AutoCloseable interface for simple cleanup using the Java try-with-resources statement, where applicable. See the *API Reference for Java (Javadoc)* for more information.
- The Apama work directory (APAMA_WORK) now has a lib directory which is on the PATH/LD_LIBRARY_PATH set up by the Apama Command Prompt (apama_env script). This lib directory can be used for user-supplied C++ correlator and IAF plug-ins.
- Client applications can now add user-defined status values which can be appended to the standard status messages of the correlator:
  - get, set and delete actions for the user-defined status values are now available from the EPL Management interface. See the corresponding section in "Using the Management interface" in *Developing Apama Applications*.
  - When the new -a (or --all) option is specified for the engine_watch tool, any user-defined status values are now appended to the standard status messages. See "Watching correlator runtime status" in *Deploying and Managing Apama Applications*.
  - User-defined status values are now visible over the REST API. See "Managing and Monitoring over REST" in *Deploying and Managing Apama Applications*.
  - User-defined status values can now be retrieved from the EngineClient API. See "EngineClient API" in *Connecting Apama Applications to External Components*.

  Note that the correlator status statements that appear in the log files will not have the user-defined status values, and will remain unaffected.
- The getMostBackedUpInput and getMostBackedUpQueueSize methods of the EngineStatus interface have changed meaning. They now give the most backed up,

or slowest, context name and queue size across all contexts, not just public contexts, as non-public channels can subscribe to specific channels and thus receive external events.

- The Scenario Browser now hides the irrelevant sections from the main panel.
  - For `Scenario`, the metadata section is hidden.
  - For `Dataview`, the input section is hidden.
  - For `Query`, the output section is hidden.
- On Linux, a Universal Messaging Docker sample is now available in the samples/docker/applications/UniversalMessaging directory of your Apama installation.
- With the previous Apama version, the correlator automatically shut down after 30 minutes when a license file could not be found. As of this version, when a license file cannot not found, Apama can be used without a time limit, but with reduced capabilities. See "Running Apama without a license file" in *Introduction to Apama*.

Platform changes in 9.10

Apama 9.10 runs on the platforms listed in the *Supported Platforms* document for version 9.10. This is available from the following web page: http://documentation.softwareag.com/apama/index.htm.

Support for the following has been removed:

- Google Chrome no longer supports Java applets. Therefore, applet-deployed dashboards will not work on Chrome browsers. WebStart and Display Server deployment will continue to work on all supported browsers.

Check the above mentioned *Supported Platforms* document for details about the newer platform versions supported by Apama 9.10. Due to these upgrades, the following versions are no longer supported:

- Software AG Terracotta BigMemory 4.3.1 (replaced by support for a more recent version).
- Software AG Universal Messaging 9.9 (replaced by support for a more recent version).
- Software AG Designer 9.9 (replaced by support for a more recent version).
- Software AG Common Runtime 9.9 (replaced by support for a more recent version).
- Oracle Java 8 update 51 (replaced by support for a more recent version).
- Google Chrome 36 (replaced by support for a more recent version).
- Firefox Mozilla 31 (replaced by support for a more recent version).
- Microsoft Internet Explorer 9 (replaced by support for a more recent version).
- Apple Safari (desktop) 5 (replaced by support for a more recent version).

Due to changes to the supported JDK version (see "Supported JDK version in 9.10" on page 24), support for the following application servers has been removed in order to support application server versions that would run using a supported JRE version:

- Oracle WebLogic Server
  - 11g 10.3.5 (still supported as a JMS provider)
  - 12c 12.1.1 (still supported as a JMS provider)
- IBM WebSphere Application Server 8.5.5.0 (still supported as a JMS provider)

Supported JDK version in 9.10

Apama is now built against Oracle JDK 8, and installs with Oracle JRE 8. JDK 7 is no longer supported.

To develop, build, and test an Apama 9.10 application, you must use Oracle JDK 8, which is the version that Software AG Designer installs. Use of any JRE other than the one that the product ships with is discouraged.

However, for applet and/or Webstart dashboards to run properly in 32-bit browsers, the latest 32-bit JREs are recommended on the client machines.

Miscellaneous changes in 9.10 affecting backwards compatibility

- With previous versions, the EPL coassignment mechanism allowed coassignments from different triggerings of a listener instance to leak into each other. For example, consider the following code:

  ```
  A a := A(0);
  B b := B(0);
  on all (A():a or B():b) {
     print a.toString() + " " + b.toString();
  }
  ```

  Previously, sending `A(1)` and `B(1)` printed the following, where the coassignment of `A(1)` carried over into the triggering of the listener caused by `B(1)`:

  ```
  A(1) B(0)
  A(1) B(1)
  ```

  This has been corrected in this version so that the above code will now print the following:

  ```
  A(1) B(0)
  A(0) B(1)
  ```

  This change may also cause a slight reduction in memory usage and an increase in performance of persistent applications.

- The `isExternal()` method on events in EPL has changed behavior. While in previous releases, the `route`, `send..to` or `enqueue..to` statements would change an event to always be internal, in 9.10, these do not modify the `isExternal()` property. For example, an external event that is forwarded to a second context will still appear to be external. The `clone()` method always returns an internal event, so if an internal event is required, then `enqueue evt.clone() to target` will give the same behavior as previous releases.
- Apama now ships with the files QueryServices.cdp and SetSingleContext.cdp instead of QueryServices.mon and SetSingleContext.mon. That is, correlator deployment packages are now provided instead of EPL files.
- The Apama extensions to the PySys testing framework now perform a clean shutdown of correlators and IAFs before the test ends, rather than terminating the process without warning. This makes it easier to detect problems associated with the shutdown, and it also allows code coverage information to be retrieved. If you want to disable this new behavior, add the boolean property `shutdownApamaComponentsAfterTest` to the project file.
- When a launch configuration is created for a project in Software AG Designer, the name "defaultCorrelator" is now used. Backwards compatibility is provided for the Apama projects that you have created with previous versions: the name "Default Correlator" is automatically considered as "defaultCorrelator".

  However, if you still have Apama projects that have been created before version 5.2 and if these projects have adapter configurations that use the `${Default Correlator.port}` and `${Default Correlator.hostname}` properties and do not have any launch configurations, the Ant export of these projects will not work seamlessly. As a workaround, you have to apply any of the following changes to these projects:

  1. When a launch configuration has been created in Software AG Designer which uses the name "defaultCorrelator", change this back to the old "Default Correlator" name.
  2. Use the new properties in your adapter configuration. That is, change the old `${Default Correlator.port}` and `${Default Correlator.hostname}` properties to `${defaultCorrelator.port}` and `${defaultCorrelator.hostname}`.
  3. Modify the Ant exported `environment.properties` file and add the value `Default Correlator.port=`*`portValue`*.
- The random number generator in the Apama correlator has been updated to provide higher resolution and improved performance. This change will result in different sequences of random numbers being produced and as such any code or tests which rely on specific sequences of random numbers via seed values (provided, for example, via the `--XsetRandomSeed` option of the `correlator` tool) or which use the "replay log" feature will need to be updated.
- Performance improvements in this version have eliminated the need for different correlator scheduler types. The `--scheduler` option has therefore been removed

from the `correlator` tool. Note that the correlator will fail to start up if you continue to use this option.

- The location of Apache Ant which is shipped with Apama has been changed and it can now be found in the third_party\apache_ant directory of your Apama installation. If you have customized any scripts to rely on the previous location, we recommend that you test your scripts carefully to ensure that they work as expected.
- The correlator now uses separate Java child classloaders for loading the libraries required for distributed MemoryStore and JMS support, which substantially reduces the number of libraries present in the top-level system classloader that all correlator, connectivity, JMS and distributed MemoryStore plug-ins share, and therefore reduces the potential for library conflicts between them and customer plug-ins. In previous versions it was possible (though not recommended) to start the correlator with distributed MemoryStore and JMS configuration files in the same directory; this is no longer possible and will result in "Failed to initialize correlator: [ADAPTER_NAME] configuration is invalid: Line XX in XML document from URL..." errors.

  To avoid "class not found" problems, ensure you are specifying classpaths using the settings in the relevant XML configuration files for JMS and distributed MemoryStore rather than using command line options that change the correlator's system classpath such as `-J-Djava.class.path`. The vast majority of applications will be doing this already and will require no modification. In some situations, if your plug-in or a library it depends upon uses the context (thread-specific) classloader to locate classes, it may also be necessary to set that explicitly using the following:

  ```
  Thread.currentThread().setContextClassLoader(
         YourClassHere .class.getClassLoader() )
  ```

Removed and deprecated features in 9.10

- Apama's Enterprise Management and Monitoring (EMM) and sentinel agent features are deprecated in this release. Apama recommends that you use Command Central instead of EMM for all deployment tasks.
- Actions without parameters that are declared as `action` *`action-name`* `{...}` are deprecated as of this version. If you still use such a declaration in your code, make sure to change this to `action` *`action-name`*`() {...}` (with additional parentheses after the action name).
- The following built-in aggregate functions are deprecated. It is recommended that you now use the alternative functions mentioned below.
  - `count(`*`value`*`)`. Use the alternative predicate aggregate function `count(not value.isNaN())` instead. Note that only the aggregate function is deprecated which uses the `decimal` and `float` values. The aggregate functions `count()` (without an argument) and `count(`*`predicate`*`)` (with a `boolean` argument) are still supported.

  - `prior(`*`value`*`,`*`index`*`)`. Use the alternative aggregate function `nth(`*`value`*`,`*`index`*`)` with a negative index instead.

- The `profiling` request of the `engine_management` tool has been deprecated. A new `cpuProfile` request, which provides the same functionality, is available instead. You can still use the deprecated `profiling` request, however, it is recommended that you now use the new `cpuProfile` request.

3 What's New In Apama 9.9

Apama 9.9 is the successor of Apama 5.3.

Apama 9.9 runs on the platforms listed in the *Supported Platforms* document for version 9.9. This is available from the following web page: http://documentation.softwareag.com/apama/index.htm. Be sure to consult that document for details about supported versions of operating systems, compilers, BigMemory, and Universal Messaging.

Apama 9.9 includes new features, enhancements, and changes as described in the following topics.

Installation changes in 9.9

As of version 9.9, Apama is installed using the Software AG Installer, and official fixes are installed using the Software AG Update Manager. See the new *Installing Apama* guide for more information.

Previous Apama releases (such as 5.3) allowed you to select one of three installation types: Server, User and Developer. In the Software AG Installer, these correspond to selecting the following options from the product selection tree:

- **Apama > Server**. Installs the server components of Apama, including the correlator and associated command line tools.

  Note: On UNIX, this is the only installation option. All other options listed here are only available on Windows.

- **Apama > Dashboard Viewer**. Installs the viewer tool that allows you to view and interact with dashboards that are receiving live data from an Apama Dashboard Server.

  Note: Unlike the old User installation type, the Software AG Installer only installs the viewer tool, and does not include any other Java/C++ libraries.

- **Designer > Apama > Apama Development Environment**. Installs Software AG Designer which provides graphical tooling for creating Apama applications.

All Apama documentation is now installed regardless of which installation type you select.

You can quickly select all Apama components (that is, the full functionality of Apama) by clicking **Typical Installations** which is shown next to the product selection tree and then selecting **Apama**.

Due to the above-mentioned changes, all Windows start menu shortcuts have changed.

The Eclipse-based development environment previously known as Apama Studio is replaced by integration into Software AG Designer. You can open Software AG Designer, which is also based on Eclipse, by choosing the following from the Windows Start menu: **All Programs > Software AG > Tools > Software AG Designer *n.n***. Note that **Software AG** is the default group name that you can change during the installation.

On Windows, the Apama work directory (`APAMA_WORK`) now defaults to the all-users `%PUBLIC%` directory, for example, C:\Users\Public\SoftwareAG\ApamaWork_*n.n* (with previous versions, it was in C:\Users\username).

On UNIX, the default for the Apama work directory is now ~/softwareag/apamawork_*n.n* .

With previous versions, the Eclipse workspace was stored under `APAMA_WORK` by default. Now it is in the location that you select when you start Software AG Designer for the first time. By default, the workspace is located in the C:\Users\username directory.

It is now also possible to install Apama using Command Central. You can then install Apama to any machine on which the Platform Manager is running. See the new *Installing Apama* guide for more information.

Query enhancements in 9.9

Apama 9.9 includes the following query enhancements:

- Queries can be configured to use the source timestamp in an event, by defining an action that returns the source time from the event's fields (performing any required transformation or parsing of the time). See *Using source time stamps of events* in *Developing Apama Applications*. Events can be marked with an annotation as not expected to arrive in order, and queries can specify an event type to use as a heartbeat to indicate that events have been delivered and are not delayed.
- Queries can contain a `with unique` clause which discards all but the latest event when many events have the same value for a specified field. See *Matching only the latest event for a given field* in *Developing Apama Applications*.
- When a number of correlators are operating in a cluster, failure of one of the correlators will now result in any timers which are active on that node being re-distributed to the other members of the cluster, so that timers are not lost. Note that Apama queries do not support reliable event delivery, so there may still be a small proportion of message loss as a result of a node failing.
- The Pysys Apama extensions now support injecting queries as part of tests.

Apama enhancements in Software AG Designer 9.9

As of version 9.9, Apama's main tool for implementing Apama applications runs in Software AG Designer, which is based on Eclipse, and is no longer called "Apama Studio".

New Apama features in Software AG Designer include:

Queries

- **Completion proposals** - completion proposals are now enabled in several parts of the Query Designer editor. This includes completion proposals in places where text editing is possible in dialogs of Filter (Where), Time From (Within), Exclusion (Without), Calculation (Select) and Filter (Having). The completion proposals are ordered by groups (variables, actions, etc.) and the items within the groups are sorted in alphabetical order.
- **Send Event action** - the channel name and the field values of the events can now be edited in a rich-text-enabled format. The string values can be edited using a mix of text and expression. The field values can now be edited in a dialog. It is now possible to add variables for expressions using a context menu.
- **Input section** - you can now configure timestamps in the Input section dialog. In the **Design** tab, the Input section table has been enhanced (additional columns for new features in this version, icon-based representation and improved tooltips), and validation markers are now shown in the corresponding parts of the dialog.
- **Error markers in table rows** - in addition to showing errors at the section level, the error markers are now shown at the row level.
- **Configure dialog** - the configure dialog allows you to manage the package (namespace) of the query and its name.
- **Dashboards for queries** - the Dashboard Generation editor can now take query files to generate the dashboard pages.

Other

- The location of the folder from which the `.ste` files are picked up has been changed to *`software_ag_install_dir`*`/Designer/extensions`.
- The launch configuration dialog for a correlator has an additional entry for referencing the Xconfig file. This supports variables in paths.
- The ANT export action no longer copies the `apama-macros.xml` file. The `build.xml` file, which is generated from the ANT export action, now references the `apama-macros.xml` file in `APAMA_HOME`.

EPL enhancements in 9.9

Apama 9.9 includes the following EPL enhancements:

- Some pre-defined annotations are now supported in EPL; see *Adding predefined annotations* in *Developing Apama Applications*. These can be used to specify default configurations for source timestamps (see *Using source time stamps of events* in *Developing Apama Applications*).
- The Time Format plug-in now has an event wrapper, which should be used in preference to directly calling the plug-in. For detailed information, see *Using the*

TimeFormat Event Library in *Developing Apama Applications* and the information on the `TimeFormat` event in the *API Reference for EPL (ApamaDoc)*.

Correlator utility enhancements in 9.9

Apama 9.9 includes the following correlator utility enhancements:

- `engine_management`

  When using the `-r` option, you should now specify distinct arguments: the request type and a list of strings. For example:

  ```
  engine_management -r profiling frequency
  ```

  In previous versions, you had to specify a single string that delimited the arguments by spaces:

  ```
  engine_management -r "profiling frequency"
  ```

  When specifying your management requests now, make sure that the arguments are no longer enclosed in quotation marks. Quotation marks are now only used for arguments that contain spaces, such as a file name. For example:

  ```
  engine_management -r setLogFile "my new log file.log"
  ```

  For more information on `engine_management`, see *Shutting down and managing components* in *Deploying and Managing Apama Applications*.

  Equivalent changes have been made to the `doRequest` methods in the Java client API and the `engine-management-request` macro in the Ant macros. Applications making use of either must be updated accordingly.

Dashboard enhancements in 9.9

Apama 9.9 includes the following dashboard enhancements:

- A new **Trends HTML5** tab is now available in the object palette. The trend charts on this tab support HTML5. The `webChartFlag` property is set to true by default.
- A new property `webLabelFlag` is now available for all objects on the **Labels** and **General** tabs of the object palette. It is also available for the checkbox object on the **Controls** tab. The `webLabelFlag` property is set to true. With this property, the display server client will render the objects from the above-mentioned tabs using HTML5 (on supported browsers) rather than in the image generated by display server. This enhancement can improve system performance in some cases and will also allow users to copy text strings from the objects to the clipboard. This property will be ignored in `dashboard_builder`, `dashboard_viewer`, Applet and WebStart clients.
- A new scalar function, `Replace All`, is now available. It replaces all occurrences of a given string which matches the pattern of the regular expression with another string.

Miscellaneous enhancements and changes in 9.9

- The Apama installation includes an Oracle Java(TM) SE Development Kit (JDK). To write and compile Java applications, you need a Java compiler such as javac, and the jar utility. Software AG recommends that you use Oracle JDK 8 included in the Apama installation to develop, build, and test your applications. The minimum JDK version you can use is JDK 7.
- Apama 9.9 incorporates ICU (International Components for Unicode) Timezone Data update 2015e, which is the most recent update at the time of release. This will update timezone data used by the correlator and TimeFormat correlator plug-in.
- The correlator will now search APAMA_WORK/license/ApamaServerLicense.xml to locate a license file if none is specified on the command line. If the license file is not found at that location, the correlator will also search Apama/etc/ApamaServerLicense.xml in the installation directory. If the found license file is expired, the correlator will fail to start up. In order to run the correlator in evaluation mode, any expired license file must be removed.

  Note: The default name for the license file is now ApamaServerLicense.xml and no longer license.txt.
- If the license cannot be found, remote connections are now allowed. You are no longer restricted to communicating only with processes on the same machine.
- The correlator log will now include the type and size of the largest container (sequence, dictionary, listener, stream window) in each `MThread` when the logging of garbage collection events has been enabled using the `verbosegc` command. This is to allow easier diagnostics of applications that are using unusual amounts of memory.
- The file `jms-mapping-spring.xml`, which stores the rules for mapping between JMS messages and Apama events, is now generated directly into the `bundle_instance_files\Correlator-Integrated_JMS` folder of an Apama project. With previous versions, this file was generated into the `bundle_instance_files\Correlator-Integrated_JMS\Generated` folder. The `Generated` folder is no longer created and used by Apama.
- The `connection` event now has a new reopen action: `reopenWithACK` which requires a callback. This is useful to know when a reopen succeeds or fails. We recommend that you use this method over the deprecated `reopen()`.
- The `shutdownConnectionOnError` action will now be based on the reconnect policy and will therefore no longer remove outstanding queries from the queue if, for example, you have a reconnect policy of `RECONNECT_AND_RETRY_LAST`. In this case, outstanding queries will remain on the queue to be retried on a successful reconnection.

New samples in 9.9 showing how to containerize Apama for Docker (Linux only)

The Docker sample directory contains configuration and samples to help you containerize and run Apama components and applications on the Docker platform, including

- Turning an Apama install into a Docker image
- Running a Docker image containing an Apama correlator, IAF adapter, Dashboard server, etc.
- Using Docker Compose to orchestrate an application running multiple connected containers

For more information about the sample, see README.txt in APAMA_HOME/samples/docker.

Platform changes in 9.9

Apama 9.9 runs on the platforms listed in the *Supported Platforms* document for version 9.9. This is available from the following web page: http://documentation.softwareag.com/apama/index.htm.

Support for the following has been removed:

- Installing Apama into a standalone Eclipse (the Apama development environment is now available only through Software AG Designer).
- 32-bit Windows platform (no longer supported even for development tasks, following its removal as a production platform in the previous release).

Check the above mentioned *Supported Platforms* document for details about the newer platform versions supported by Apama 9.9. Due to these upgrades, the following versions are no longer supported:

- Software AG Terracotta BigMemory 4.3.0 (replaced by support for a more recent version).
- Software AG Universal Messaging 9.8 (replaced by support for a more recent version).
- Software AG Common Runtime 9.8 (replaced by support for a more recent version).
- Oracle Java 8 update 25 (replaced by support for a more recent version).
- Microsoft SQL Server 2008 R2 (replaced by support for a more recent version).
- Apache Tomcat 7.0.53 (replaced by support for a more recent version).
- Google Chrome below version 36 (replaced by support for a more recent version).

Renaming of Apama client libraries in 9.9

Several libraries have been renamed. In most cases, the version number has been removed from the file names and an "ap" prefix has been added. Due to this change, you may need to update your build scripts and automated tests. Software AG Designer will automatically upgrade any Apama project with the "Apama Java" nature to have the renamed jar files on its build path.

The following table lists the most important name changes:

Old name	New name
Windows:	
apcommon5.3.lib	apcommon.lib
engine_client5.3.dll	apclient.dll
engine_client5.3.lib	apclient.lib
engine_client_dotnet5.3.dll	apclientdotnet.dll
iafcore5.3.lib	apiaf.lib
Linux:	
libengine_client.so	libapclient.so (-lapclient)
Jar files:	
dashboard_client5.3.jar	ap-dashboard-client.jar
correlator_extension_api5.3.jar	ap-correlator-extension-api.jar
engine_client5.3.jar	ap-client.jar
jplugin_public5.3.jar	ap-iaf-extension-api.jar
util5.3.jar	ap-util.jar
distmemstore-bigmemory.jar	ap-distmemstore-bigmemory.jar

Old name	New name
distmemstore5.3.jar	ap-distmemstore.jar

Note: You should no longer reference `dashboard_studio`*n.n*`.jar`. All required classes are now available in `ap-dashboard-client.jar`.

Miscellaneous changes in 9.9 affecting backwards compatibility

- Correlator persistence can now only be enabled with a valid license file. Otherwise, it will not be possible to run the correlator in persistent mode.
- Apama supports various licensing models, one of which is licensing per physical CPU core. In previous releases, any restrictions on the number of licensed cores were not enforced. In this release, the number of threads that can be used for running Apama applications will be capped at the number of cores specified in the license, if the correlator is run on a machine that has more cores than the license allows. The correlator will log a `WARN` message to make it clear when this has happened. If you have previously been running with an insufficient license, then you may experience a corresponding reduction in performance after upgrading.
- The Ant macro `<start-correlator>` (generated by the **Ant Export** feature) checks whether there is already a correlator running. However, previously that was ignored allowing `<start-correlator>` to succeed even if a new correlator with the specified parameters was not started. Now it will log a warning and (by default) fail the task if a correlator is already running. An Ant build script exported from a launch configuration with **Reuse Correlator** selected will therefore now fail. To return to the previous behavior, edit the `build.xml` to set the `failoncorrelatorrunning=false` attribute on the `<start-correlator>` task, or set the global property `correlator.failonrunning`, for example, by specifying `-Dcorrelator.failonrunning=false` on the command line when executing Ant.
- When you upgrade to a new major version of the product, you must regenerate any CDP that was created by exporting query files or scenario definitions from the previous version.

Removed and deprecated features in 9.9

- The Log File Manager plug-in (`LoggingManager`) is being deprecated and should not be used any more, as it will be removed in a later release. In the future, you should just use the EPL `log` statement to write log messages, and configure logging using the Management interface (see *Using the Management interface* in *Developing Apama Applications*). New actions have been added to the `com.apama.correlator.Logging` event, such as `setApplicationLogFile`, `setLogFile` and `setApplicationLogLevel`. These actions are the equivalent of using the `engine_management` requests to configure logging (see also *Shutting down and managing components* in *Deploying and Managing Apama Applications*). In previous releases `com.apama.correlator.Logging` had only one action on it, namely `rotateLogs()`.

- Apama no longer provides ODBC drivers. Please use your own ODBC drivers if you need to use ODBC. Please note that Software AG recommends using JDBC rather than ODBC with Apama.
- The ADBCHelper actions `findAvailableServers` and `findAvailableServersFull` have been deprecated, and replaced with more appropriately named `findAvailableDataSources` and `findAvailableDataSourcesFull` respectively. These are identical in behavior.
- The `reopen` action on the connection event of `ADBCEvents.mon` has been deprecated. Please use the action `reopenWithAck()` instead.

4 What's New In Apama 5.3

Apama 5.3 runs on the platforms listed in the *Supported Platforms* document for version 5.3. This is available from the following web page: http://documentation.softwareag.com/apama/index.htm. Be sure to consult that document for details about supported versions of operating systems, compilers, BigMemory, and Universal Messaging.

Apama 5.3 includes the following new features, enhancements, and changes.

New Apama queries capability

The new Apama queries capability lets business analysts and developers create scalable applications that process events originating from very large populations of real-world entities. 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. This makes it easy to deploy across multiple servers, and keep the application running even if some servers are taken down for maintenance or fail. Apama queries can be used alongside EPL monitors in the same correlator process, interacting by sending events between them.

Incoming events that queries process are partitioned by, for example, customer account numbers, car license plate numbers, devices or some other entity. In a query application, the correlator processes the events in each partition independently of other partitions.

Advantages of Apama queries over Apama monitors:

- When used in conjunction with BigMemory, queries provides active-active availability. That is, queries can be run in a cluster, where every node in the cluster contributes processing resources. The number of nodes can be changed dynamically without losing state.
- Queries can be run on correlator clusters.

Disadvantages of Apama queries compared to Apama monitors:

- Higher latency than monitors. Latency is of the order of milliseconds to seconds rather than microseconds to milliseconds. Exact values depend on the deployment and the types of events being processed.
- Apama monitors allow you to write custom and more powerful EPL applications that do not have the declarative and structural bounds that queries have.

To take advantage of the scalability and availability that the queries platform offers, the problem your application needs to solve should meet one or more of the following requirements:

- Different partitions for a given query must be completely independent. However, different queries can use different partition keys for the same event types. For example, one query may partition ATM withdrawals by `cardNumber`, and another by `atmId`.

- The average number of events in each event window in a partition is low. The recommendation is less than 50 events. For example, if ATM withdrawals are partitioned by `cardNumber` then a window that retains withdrawals for a three-day period is fine because the typical number of withdrawals per card is likely to be low. While it is possible to have hundreds of withdrawals for a single card number, that would be an exceptional case and probably indicative of suspicious behavior.
- Other than the history of events, no state is required. Queries do not provide for state to be stored. However, it is possible to mix monitors and queries in the same deployment.
- The time between events destined for the same partition is typically long, that is, more than a few seconds between events.
- The exact ordering between events is not critical. A query may treat two events for the same partition that occur close in time as having occurred in an order that is different from the order in which they were sent.

Apama queries are designed to be easy to develop for both the business analyst and the application developer. Graphical tools to specify the application design and full round-trip engineering allow both the business analyst and the developer to work on the same queries. At the developer level, an Apama query is defined using the Apama event processing language, EPL. A new channel, `com.apama.queries`, is provided for sending events to queries. To implement queries, Apama provides:

- The new Query Designer editor in Apama Studio. This graphical user interface lets business analysts define queries without the need to write code.

  See "Adding query files to projects" in *Using the Apama Studio Development Environment* (renamed to *Using Apama with Software AG Designer* in version 9.9).
- New EPL constructs for application developers to write code that defines queries. A query is defined in a `.qry` file. A query cannot contain a monitor and a monitor cannot contain a query. A query can use many, but not all EPL constructs.

  To use EPL to define a query, see "Defining Queries" in *Developing Apama Applications*.
- Information and instructions for configuring, deploying and managing queries. To scale query deployment across multiple machines, you use Apama's distributed MemoryStore with a JMS bus such as Universal Messaging.

  See "Deploying and Managing Queries" in *Deploying and Managing Apama Applications*.

Apama Studio enhancements in 5.3

The main enhancements to Apama Studio support the implementation of Apama query applications. New Apama Studio features include:

- Query Designer editor offers two views for defining and updating query definitions:

  - **Design** — Business analysts define and update query definitions in this view. The **Design** tab provides an event palette and a canvas for defining an event pattern of interest. Toolbars, tooltips, and dialogs make it easy to define the events of interest to the query as well as any required parameters, conditions for finding matches, aggregate calculations and actions to take when a match set is found.
  - **Source** — Application developers can define and update query definition source code in the EPL code editor available from the **Source** tab.
- Scenario Browser has been enhanced so that you can now use it to do the following:
  - Create new instances of queries by specifying query parameter values.
  - Change the parameter values for a query that is running.
  - Monitor query activity in the local correlator.
- EPL Objects view: A new view that appears beside the outline view in the Apama developer perspective that shows logical view of Apama objects of an Apama project in a tree structure. The view presents monitors, events and aggregates available in the Apama project. It also has options to filter the tree content and group by.
- The Apama Workbench perspective default behavior is that it shows query and event files in addition to dashboard and scenario files.
- Exporting an Ant deployment script includes all files needed to deploy queries on a correlator host machine.
- Exporting a project to a correlator deployment package (CDP) can include any query files in your project.
- A default filter with value `com.apama.*` has been added to the Event Chooser dialog to automatically remove events in the `com.apama` namespace from the event list.
- All events starting with __ (two underscore characters) will be treated as internal events and automatically filtered from the Event Chooser, EPL Query, and EPL Objects View dialogs.

Log rotation enhancements in 5.3

When you specify the name of any of the following log files:

- Correlator status log
- Correlator input log
- Any application log files

You can now use one, two or all three of the following string substitutions in any order:

```
${START_TIME}

${ROTATION_TIME}

${ID}
```

In the previous release, string substitutions were allowed for only the name of the input log file. Also, this release adds the `${ROTATION_TIME}` string substitution. For example, to specify an application log filename for messages generated in `com.example.mypackge`, you could specify the log filename as follows:

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

The format is as follows:

```
file[one_or_more_substitutions_in_any_order].log
```

`file`	Replace *`file`* with the name of the file that you want to be the log file. If you also specify `${START_TIME}` and/or `${ROTATION_TIME}` and/or `${ID}`, the correlator prefixes the name you specify to the time the correlator was started and/or the time the log file was rotated (logging to a new file began) and/or an ID, beginning with `001`.
`${START_TIME}`	Tag that indicates that you want the correlator to insert the date and time that it started into the log filename.
`${ROTATION_TIME}`	Tag that indicates that you want the correlator to insert the date and time that it starts sending messages to a new log file into the log filename.
`${ID}`	Tag that indicates that you want the correlator to insert a three-digit ID into the filename of the log file. The ID that the correlator inserts first is `001`. The log ID increment is related only to rotation of log files. The ID allows you to create a sequence of log files. Each time the log file is rotated, the correlator increments the ID.

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

Apama 5.3 provides two new interfaces for rotating all Apama log files. Log file rotation means that the log file in use is closed and a new log file begins to be used. Each of the following new interfaces rotates the correlator status log, correlator input log if the correlator is generating one, and any application log files.

- The `engine_management` utility now takes the following option:

  ```
  engine_management -r rotateLogs
  ```

- In an EPL monitor that uses the Management interface correlator plug-in, you can now call the `rotateLogs()` action. The `rotateLogs()` action is defined in the `com.apama.correlator.Logging` event.

For details, see "Rotating the correlator log file" and "Rotating all log files" in *Deploying and Managing Apama Applications*.

EPL enhancements in 5.3

Apama 5.3 includes the following EPL enhancements:

- You can now use `dictionary` and `sequence` literals anywhere in an EPL program and not just in variable initializations as in previous releases. For example, the following code fragments are now valid EPL:

  ```
  setMapping({"SOW":"Software AG", "MSFT":"Microsoft"});
  route FooBar(true, [100.0, 1000.0, 100000.0]);
  ```

- A new channel is provided. To send an event to all running Apama queries, send the event to the `com.apama.queries` channel.

Correlator-integrated messaging for JMS enhancements in 5.3

Enhancements to correlator-integrated messaging for JMS include the following:

- APIs for implementing custom mapper

  Apama now provides APIs for implementing your own Java class for mapping between Apama event strings and JMS message objects. If the mapping tools provided with Apama do not meet your needs you can use these APIs to create a custom mapper. A custom mapper can handle some event types and delegate handling of other event types to the mapping tools provided with Apama or to other custom mappers. Typically, you will want to use the `SimpleAbstractJmsMessageMapper` class, which is in the `com.apama.correlator.jms.config.api.mapper` package.

- Apama mapping tools support custom resolvers and functions

  The mapping tools for JMS that are provided with Apama have been enhanced to include support for custom resolvers.

- New `APP_CONTROLLED` reliability mode

  Apama now supports JMS `APP_CONTROLLED` reliability mode for receivers. This reliability mode allows applications to implement no-loss messaging without using correlator persistence.

- New support for applications to send messages reliably without using correlator persistence. An application can request `BEST_EFFORT` senders to flush messages to the JMS broker. Applications are notified after all messages already sent to the JMS sender channel have been sent to the broker. The application defines a strategy for

  - Preserving state that allows re-generation of messages in case of a correlator failure.

    - Cleaning up preserved state after receiving an acknowledgement that all messages sent to a JMS receiver channel have been flushed to the JMS broker.
- JUEL mapping expressions that can be used to get or set elements of a JMS message now include resolver expressions for obtaining sender, receiver, and connection IDs.

Reference information for the APIs is provided in Javadoc format. Introductory and general information is in *Connecting Apama Applications to External Components* in the following topics:

- "Using a custom Java mapper"
- "Using custom mapping extensions in correlator-integrated adapters for JMS"
- "Receiving messages with `APP_CONTROLLED` acknowledgements"
- "Sending messages reliably with application flushing notifications"
- "JUEL mapping expressions reference for JMS"

Dashboard enhancements in 5.3

A dashboard trend graph object can now be rendered by using HTML5, which provides added functionalities to the chart object without requiring a Flash Player or other browser plug-in. To enable this feature, you select the `webChartFlag` option from the list of trend graph properties.

This feature applies only to display sever dashboard deployment.

When `webChartFlag` is selected some minor trend graph properties are not supported, there are two new properties, and there are some differences in behavior. For details about using trend charts when `webChartFlag` is selected, see "Rendering trend charts in HTML5" in *Building and Using Dashboards*.

Killing an object now requires specification of a monitor name

In all client APIs (Java, C++, C and .NET), killing an object now requires specification of a monitor name. In previous releases, specification of any name (for example, package name, event name, aggregate name) was allowed but the result was unpredictable. Apama 5.3 makes the behavior consistent and predictable. The APIs that now require the name of a monitor are:

Java	`DeleteOperationsInterface.killName()`, `killNames()` and `killNamesFromFile()`
C++	`EngineManagement.killName()`

C	`AP_EngineManagement_Functions.killName()`
.NET	`ICorrelatorManagement.KillName()`, `KillNames()` and `KillNamesFromFile()`

Execution of the `engine_delete` utility with the `-k` option, which kills the specified object, also now requires specification of the name of at least one monitor. For example:

```
engine_delete -k com.mycompany.MyOrderMonitor
```

Execution of `engine_delete` with the `-k` option kills every instance of the specified monitor(s) whether or not the instance is processing anything. Any `ondie()` and `onunload()` actions defined in killed monitors are not executed.

New sample project showing how to manage Universal Messaging DataGroups

A new Apama sample project shows how Apama can

- Publish to Universal Messaging (UM) DataGroups
- Dynamically manage the membership of UM DataGroups as clients, known as DataStreams, connect and disconnect to the UM server.

UM DataGroups use a publish/subscribe messaging technology, which has a lightweight grouping structure that allows a remote process to manage the data received by each client.

Note: UM DataGroups use a completely different mechanism than the one used by JMS topics or UM channels, which can be accessed through Apama's correlator-integrated messaging for JMS or through UM native integration.

The new sample has two parts:

- A suggested EPL event API to expose the DataGroup publisher and manager functionality. It is implemented using a correlator plug-in that is written in Java. Complete source code is provided.
- A sample application that shows simple and advanced ways to use this API for publishing and/or managing DataGroups. This demonstrates the powerful combination of the Apama platform and the UM DataGroups feature.

For more information about the sample, see `README.txt` in `APAMA_HOME/samples/universal_messaging/datagroups`. For more information about UM DataGroups, refer to the Universal Messaging documentation.

C++ and C Correlator plug-in API enhancements

Correlator plug-ins written in C++ and C can now return sequences. The APIs for developing correlator plug-ins in C++ and C have been enhanced as follows:

- The C++ API class `AP_Type` now defines the following functions:
  - To set the length of an existing sequence:

    ```
    void setSequenceLength(size_t length)
    ```

  - To create a sequence that contains empty items of the type you specify:

    ```
    void createSequence(AP_TypeDescriptor inner)
    ```

  Also, `AP_Type` objects can now be directly assigned from one to another. For example, to return the first argument, regardless of type:

  ```
  rval = args[0];
  ```

- The C API class `AP_PluginType_Functions` now defines the following comparable functions:
  - `void (AP_PLUGIN_CALL *setSequenceLength)(const AP_PluginType *obj, AP_uint32 len)`
  - `void (AP_PLUGIN_CALL *createSequence)(const AP_PluginType *obj, AP_TypeDescriptor inner)`
  - `void (AP_PLUGIN_CALL *copyFrom)(const AP_PluginType *obj, const AP_PluginType *other);`
- For both C++ and C APIs, a sequence you create can contain items of type `integer`, `float`, `boolean`, `string`, or `chunk`.

For additional information, see "Working with sequences" in "Developing Correlator Plug-ins", which is part of *Developing Apama Applications*.

Miscellaneous enhancements and changes in 5.3

It is now possible to specify the Java classpath individually for each JMon application or Java plug-in, using a new `classpath` element in the jar file's deployment descriptor XML or `@Application` annotation. See "Specifying classpath in deployment descriptor files" in *Developing Apama Applications*.

When you start a correlator and specify the `-J` option you can now specify `-J-Djava.class.path=`*`path`* and the path you specify will be appended to all internal Apama `JAR` files. In previous releases, Apama used only the setting of the `CLASSPATH` environment variable. If you specify both the `CLASSPATH` environment variable and a classpath on the command line then the classpath specified on the command line takes precedence.

The MATLAB plug-in has been moved from the Apama Capital Markets Foundation into Apama 5.3. This plug-in includes the MATLAB bundle, libraries and executables that let Apama applications use MATLAB products. Documentation for using MATLAB products in an Apama application is now in *Developing Apama Applications*, *Developing Apama Applications in EPL*, "Using Correlator Plug-ins in EPL", "Using MATLAB products".

The Apama extensions for the Pysys testing framework now include `injectCDP()` method to allow a CDP to be injected into a correlator from within a test.

The event parse error logged by the correlator now contains the character offset of the error and the field member name that was being parsed. The expected event format logged is now recursive so nested event fields are also fully logged.

Apama 5.3 incorporates ICU (International Components for Unicode) Timezone Data update 2014j_44, which is the most recent update at the time of release. This will update timezone data used by the correlator and TimeFormat correlator plug-in.

Platform changes in 5.3

Apama 5.3 runs on the platforms listed in the *Supported Platforms* document for version 5.3. This is available from the following web page: http://documentation.softwareag.com/apama/index.htm.

ADBC databases are now supported only using the drivers included with Apama (except for MySQL, which is supported using MySQL's driver). Use of the JDBC drivers is strongly recommended in all cases, and the use of ODBC drivers (those supplied with Apama and the MySQL ODBC driver) is deprecated in this release.

The 32-bit Windows platform is now supported only for development tasks. Production use of Apama is not supported on this platform.

Check the Apama 5.3 Supported Platforms document for details about the newer platform versions supported by Apama 5.3. Due to these upgrades, the following versions are no longer supported:

- Software AG Terracotta BigMemory 4.1.4.
- Software AG Universal Messaging 9.7 is no longer supported, for either JMS or native Apama messaging.
- Oracle Java 7 (update 25 and higher) is supported for only client-side (client API applications, dashboards) use. Oracle Java 6 is no longer supported.
- MySQL 5.1.35
- IBM DB2 9.5.
- HornetQ 2.3.0.
- Eclipse 4.3.
- Mozilla Firefox below version 31.

- Microsoft Internet Explorer below version 9.
- Apple Safari desktop below version 5. (NB: the supported Safari version on the iPad is unchanged).
- WebSphere Application Server 7.0.0.13.

JDK 7 now minimum JDK version supported

Apama is now built against JDK 7, and installs with JRE 8.

You can no longer use JDK 6 (or older) to develop Apama applications nor can you use JRE 6 (or older) to deploy Apama applications. While it should not be necessary to rebuild applications previously built with an older Java version, the recommendation is that you build your application with the newer JDK for two important reasons:

- To ensure that you detect compatibility problems before you want to deploy the application
- To take advantage of performance improvements in newer Java compiler versions

To develop, build, and test an Apama 5.3 application, the recommendation is that you use Oracle JDK 8. The minimum you can use to build your application with Apama 5.3 libraries is JDK 7.

To deploy an Apama 5.3 application, the recommendation is that you use Oracle JRE 8, which is the version that Apama installs. Use of any JRE other than the one that Apama ships with is discouraged.

With Apama 5.3, web application servers and client web browsers must be upgraded to JRE 7 or later to access WebStart or Applet dashboards (JRE 8 is recommended). This requirement is due to the Java compiler changes in Apama 5.3.

Removed and deprecated features in 5.3

The following requests have been removed from the `engine_management` utility:

- `-r reopenLog`
- `-r rotateInputLog`
- `-r rotateReplayLog`

Instead, specify the `-r rotateLogs` request. See "Log rotation enhancements in 5.3 " on page 42.

The ODBC drivers included with Apama are deprecated in this release. Use the provided JDBC drivers.

The JDBC-ODBC bridge has been removed due to the Java 8 upgrade. Use the provided JDBC drivers.

Dashboards no longer support ODBC as a database source, due to the JDBC-ODBC bridge removal. The "Use ODBC Driver" option is enabled by default, this must now be disabled and the JDBC driver used.

The Visual Event Mapper is no longer available for ODBC data sources, due to the JDBC-ODBC bridge removal. The ODBC adapter Editor will no longer show the mapping tab to map events and generate supportive monitor script.

The event action name `getSourceTime` is now reserved for future use.

Backwards incompatibility with persisted projects recovered to 5.3 from older versions

To support the new Queries feature in Apama 5.3, some internal (non-public) event definitions have been changed in `ScenarioService.mon`. The change will have no effect on most applications as this API is not officially supported or documented. However it may affect customers using a persistent correlator to maintain application state across an upgrade from Apama 5.2 (or earlier) to 5.3 (or later), if they also make use of scenarios, DataViews or the MemoryStore plug-in.

This is because the EPL that is recovered from the persistent store after the upgrade will use the older (pre-5.3) event definitions and these are not compatible with post-5.3 Scenario Service clients and dashboards, or with the MemoryStore plugin. As a result users may see event parsing failures logged by the correlator, and/or Scenario Service clients and dashboards reporting errors such as *ScenarioService cannot parse event due to mismatch between the EPL event definition and the ScenarioService client* .

In most cases it should still be possible to recover from a pre-5.3 persistent store without losing application state, provided the following actions are taken immediately after starting the correlator post-upgrade:

- If using DataViews and the `DataViewService_Impl_Dict.mon` monitor, delete it using `engine_delete com.apama.dataview.DataViewService_Impl_Dict`
- If using MemoryStore and the `MemoryStoreScenarioImpl.mon` monitor, delete it using `engine_delete com.apama.memorystore.MemoryStoreScenarioImpl`
- Delete any application Scenario definitions, using `engine_delete Scenario_`*`SCENARIO_NAME`*

  Note that Scenario instances are never persistent so the state of any running Scenario instances would not be preserved across a correlator restart anyway.
- Delete the Scenario Service EPL using `engine_delete com.apama.Scenario`
- Re-inject the latest version of the Scenario Service from `APAMA_HOME/monitors/ScenarioService.mon`
- If using `MemoryStoreScenarioImpl` (see above), reinject the latest version from `APAMA_HOME/monitors/data_storage/MemoryStoreScenarioImpl.mon`

- If using `DataViewService_Impl_Dict` (see above), reinject the latest version from `APAMA_HOME/monitors/DataViewService_Impl_Dict.mon`
- Re-inject any Scenario definitions that are part of your application
- Send any events that are required to reinstantiate Scenario instances

5 What's New in Apama 5.2

Apama 5.2 runs on the platforms listed in the *Supported Platforms* document for version 5.2. This is available from the following web page: http://documentation.softwareag.com/apama/index.htm. Be sure to consult that document for details about supported versions of operating systems, compilers, BigMemory, and Universal Messaging.

Apama 5.2 includes the following new features, enhancements, and changes.

Apama Studio enhancements in 5.2

Apama 5.2 adds the following enhancements to Apama Studio.

Apama Studio now uses Eclipse 4.3.

When Apama Studio starts it now collects any files in Apama's new `studio\extensions` folder and uses those files to add resources to the Apama Studio environment. This is useful when you want to import projects that have dependencies such as environment variables or catalogs of blocks, bundles or functions. In previous releases, when you imported projects that had such dependencies you manually added each resource before you could build the project.

See "Setting up the environment before importing projects" in *Using the Apama Studio Development Environment* (renamed to *Using Apama with Software AG Designer* in version 9.9).

There are several enhancements for creating launch configurations:

- In the **Run Configurations** dialog, when you select the **Common** tab, the default selection for **Save as** is now **Shared file** with a default path of `*project_name*\config\launch`. In previous releases, **Local file** was selected by default.

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

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

  In previous releases, creating a launch configuration generated only a `.launch` file, which contained Apama information but was managed by Eclipse. Now Apama manages the content of the `.deploy` file.

  Any changes you make to the launch configuration will be reflected in the `.deploy` file.

- When you export an Ant launch configuration and you select **Generate initialization list during launch**, Apama Studio uses the `.launch` and `.deploy` files to create the deployment script. The Ant deployment script then uses `.deploy` file at runtime to read the injection order details.

- In the **Run Configurations** dialog, in the **Components** tab, a new **Connections** button is available. After you add two or more correlators, click **Connections** to specify connections between correlators. This has the same effect as using the `engine_connect` correlator utility to connect correlators.

  The Ant export facility uses this connection information to create the required `engine_connect` tasks in the deployment script. Any `engine_connect` options that you specify in the **Connections** tab are saved in the `.deploy` file.

- In the **Correlator Configuration** dialog, the **Initialization** tab has been rearranged into two tabs as follows:

  - The **Injections** tab lists the files to be injected except for event files. The files will be injected in the order in which they appear in the **Injections** tab. You can accept the default **Automatic Ordering** calculated by Apama Studio or you can select **Manual Ordering** to change the default order. If you change the default order then it is up to you to ensure that you specify a correct injection order. You can also de-select a file so that it is not injected in a particular launch configuration. The file remains in the project.

  - The **Event Files** tab lists the `.evt` files to be injected. You can de-select a file so it is not injected in a particular launch configuration. You can select a `.evt` file and move it up or down in the list. The order of the files in the list on the **Event Files** tab is the order in which they will be injected. The default behavior is that Apama Studio lists the files in lexicographical order.

  The information in the `.deploy` file accommodates any injection order changes you make in the **Injections** tab or **Event Files** tab of the **Correlator Configuration** dialog.

  In Apama 5.2, EPL files are injected first and then `.evt` files are injected. In previous releases, EPL files and `.evt` files were injected in the order in which they appeared in the **Initialization** tab, which meant that they might be interspersed. This might cause a backward incompatibility for launch configurations created with previous releases.

New feature for addressing contexts with channels in 5.2

Apama 5.2 extends the ability to send events to named channels by providing a more powerful and general feature for using channels. In your application, each context, external receiver, correlator plug-in, or Apama client application can subscribe to one or more channels to receive the events sent on that channel. Adapter configurations can also now specify the channels on which the adapter sends events to correlators.

When a monitor instance subscribes to a channel it receives events delivered to that channel from adapters and other contexts. All monitor instances in the same context as the subscribing monitor instance receive the events delivered to the subscribed channel. Within a context, there can be subscriptions to different channels. The delivery of events from adapters, over multiple channels, to multiple contexts, is parallel from adapter to processing context, and does not impose any serial bottlenecks.

To achieve the best performance, the way in which data is organized for delivery from adapters to multiple channels must be determined. Typically, this organization matches the way you distribute the work of your application into contexts. Data sent to a particular channel should have no ordering dependency on data sent to any other channel.

The new channel features include the following enhancements and changes as well as additions to Apama Studio that support the use of channels.

EPL

- The new `send...to` statement sends the specified event to the specified channel. Use the `send...to` statement in place of the `enqueue...to` or `emit..to` statement. The `enqueue...to` and `emit..to` statements will be removed in a future release.
- The new `monitor.subscribe()` and `monitor.unsubscribe()` statements subscribe/unsubscribe a context to the specified channel. The monitor that contains the statement and any other monitors in the same context are subscribed/ unsubscribed. Note that this is a special use of the `monitor` keyword.
- The new `com.apama.Channel` type holds a string or a context. You can send events to `com.apama.Channel` objects. If the `Channel` object contains a string then the event is sent to the channel of that name. If the `Channel` object contains a context then the event is sent to that context.

Correlator utilities

- The `engine_send` utility and `engine_receive` utilities can send/receive a file that associates events with channels. For details, see "Event association with a channel" at the end of the *Correlator Utilities Reference* section of *Deploying and Managing Apama Applications*.
- The `engine_connect` utility has a new parallel mode (`-m parallel` or `--mode parallel`) that uses a connection per channel between correlators and delivers events to the target correlator on the channel the events were sent from in the source correlator. Previous behavior of `engine_connect` is preserved in legacy mode (`-m legacy` or `--mode legacy`), which uses one connection between two correlators and delivers all events to the default channel.

  All connections to the correlator are now persistent. Consequently, the `engine_connect` utility no longer needs to provide the `--persistent` option and this option has been removed.
- The `engine_inspect` utility now provides information about channels being used and about receivers.
- When the correlator sends status to its log it now provides information for the following new status indicators:
  - `srn` — Slowest receiver name is the name of the receiver whose queue has the largest number of entries. If no receivers have queue entries then this value is `"<none>"`.

- `srq` — Slowest receiver queue. For the receiver identified by `srn`, the slowest receiver, this is the number of entries on its queue.

Apama client APIs

In C, C++, Java and .NET, there is a new enumeration called `ConnectMode` with values `CONNECT_LEGACY` or `CONNECT_PARALLEL`. This is used by new overloadings for the existing methods `attachAsEventConsumerTo()`, `detachAsEventConsumerFrom()`, `attachAsConsumerOfEngine()` and `detachAsConsumerOfEngine()`.

There is now a separate connection for each `EngineManagement` object (`EngineClientBean` object, `EventService` object, or `ScenarioService` object) that you create in a client application. In previous releases, multiple `EngineManagement` objects shared a connection to the same Apama component.

For events that have a channel attribute set, the value of that attribute is now used when the event is sent to a correlator. In previous releases, the value of a channel attribute was ignored. For events that do not have a channel attribute set, the behavior is unchanged. That is, the event is delivered on the default channel (the empty string) to all public contexts.

The `setEngineParams()` method no longer accepts the `LogicalID` argument.

Correlator plug-in APIs

- Correlator plug-ins must be rebuilt against the new header files.
- When writing a correlator plug-in in C++, the following new methods are defined by `AP_CorrelatorInterface`:

```
virtual void sendEventTo(const char* event, AP_uint64 targetContext,
   AP_uint64 sourceContext)
void subscribe(const AP_EventHandlerInterface::ptr_t &handler,
   std::initializer_list<const char *> channels)

template<typename ITER>
   void subscribe(const AP_EventHandlerInterface::ptr_t &handler,
   const ITER &start, const ITER &end)

void subscribe(const AP_EventHandlerInterface::ptr_t &handler,
   const T &channel)

void unsubscribe(const AP_EventHandlerInterface::ptr_t &handler,
   std::initializer_list<const char *> channels)

template<typename ITER>
   void unsubscribe(const AP_EventHandlerInterface::ptr_t &handler,
   const ITER &start, const ITER &end)

void unsubscribe(const AP_EventHandlerInterface::ptr_t &handler,
   const T &channel)

virtual void unsubscribe(const AP_EventHandlerInterface::ptr_t &handler)
```

- When writing a correlator plug-in in Java, the following new methods are available on `com.apama.epl.plugin.Correlator`

```
public static native void sendTo(String evt, String chan);
public static native void sendTo(String evt, Context ctx);
public static native void sendTo(String evt, Context[] ctxs);

public static native void subscribe(EventHandler handler, String[] channels);
public static native void unsubscribe(EventHandler handler, String[] channels);
public static native void unsubscribe(EventHandler handler);
```

Adapter configurations

- When specifying connections to correlators, you can specify the optional `parallelConnectionLimit` attribute in a `<sinks>` element. Normally, you do not need to specify this attribute. The default behavior is that the IAF limits itself to an internally set number of connections with each specified sink. This number scales according to the number of CPUs that the IAF detects on the host that is running the IAF. While this number is usually sufficient, there are some situations in which you might want to change it. For example, if you are trying to conserve resources you might want to limit the number of connections to 1, or if you want to prevent multiple threads from sharing a connection you might allow a higher number of connections than the default allows.
- When specifying the mapping between an Apama correlator event type and a kind of external message, you can now specify the attributes `transportChannel` and/or `presetChannel` to support getting and setting the channel of each normalized event that passes through the IAF adapter.

  For details, see "Event mappings configuration" in the *Developing Adapters* part of *Developing Apama Applications*.

Application upgrade considerations arising from channels improvements

After you install Apama 5.2, you will need to make some modifications to your applications to take advantage of the new channel features. Before you make these modifications, consider the following:

- Does your application depend on event ordering? Applications that use multiple channels might re-order events.
- How many channels do you need? Which events would use which channels? What is your naming convention for channels? Typically, the way your application organizes work into contexts is a good indicator for how to use channels.
- In multi-correlator deployments, can you use fewer correlators? By using channels, a correlator provides greater parallel processing, which might render some additional correlators unnecessary. Using fewer correlators reduces the complexity of your deployment and can improve performance.
- Can you use Universal Messaging rather than the `engine_connect` utility? This also reduces the complexity of your applications and allows you to benefit from other UM features. See "New feature for using Universal Messaging to connect Apama components" on page 59.

To start using channels:

- In EPL monitors, `subscribe` to receive events delivered to particular channels. Also, use `send` statements to send events to particular channels.
- In adapter configurations, when specifying the mapping between an Apama correlator event type and a kind of external message, specify new channel attributes in `<event>` and `<unmapped>` elements. See "New feature for addressing contexts with channels in 5.2" on page 55 for details.
- In correlator plug-ins, use the new methods for subscribing to receive events sent on particular channels and for sending events to particular channels. These methods are provided by:
  - C++: `AP_CorrelatorInterface`
  - Java: `com.apama.epl.plugin.Correlator`

To upgrade multi-correlator deployments, the additional recommended steps are:

- In `engine_connect` command lines, specify the `-m parallel` or `--mode parallel` option. In parallel mode, for each specified channel, Apama will create a separate connection between the sending and target correlators.
- In event files to be sent by the `engine_send` utility, specify channels. For details, see "Event association with a channel" at the end of the *Correlator Utilities Reference* section of *Deploying and Managing Apama Applications*.
- In the parts of your application that receive output from the `engine_receive` utility, make any changes needed to accommodate the new channel specification associated with events.

New feature for using Universal Messaging to connect Apama components in 5.2

Universal Messaging (UM) is Software AG's middleware service that delivers data across different networks. It provides messaging functionality without the use of a web server or modifications to firewall policy.

Previous Apama releases provided access to UM only through the correlator-integrated adapter for JMS. With Apama 5.2, there is support for Apama applications to directly use UM to transport events between Apama components. This can make deployments more flexible and can simplify configuration.

In an Apama application, correlators and adapters can connect to UM realms or clusters. A correlator or adapter connected to a UM realm or cluster uses UM as a message bus for sending Apama events between Apama components. Connecting a correlator or adapter to UM is an alternative to

- Defining connections between an adapter and particular correlators in the `<apama>` element of an adapter configuration file

- Specifying a connection between two correlators by executing the `engine_connect` correlator utility

Using UM can simplify an Apama application configuration. Instead of specifying many point-to-point connections you specify only the address (or addresses) of the UM realm or cluster. Apama components connected to the same UM realm use UM channels to send and receive events. (UM channels are similar to JMS topics.)

When an Apama application uses UM a correlator can be configured to automatically connect to the required UM channels. There is no need to explicitly connect UM channels to individual correlators. A correlator automatically receives events on UM channels that monitors subscribe to and automatically sends events to UM channels.

You can use UM only to send and receive events. You cannot use UM for EPL injections, delete requests, engine send, receive, watch or inspection utilities, or `engine_management -r` requests.

Only UM channels can be used with Apama. UM queues and datagroups are not supported in this Apama release.

In Apama Studio, you can add UM support to a project. After you do that, you can choose whether to specify UM messaging in correlator launch configurations and in adapter configurations.

In an IAF adapter configuration file, the new `enabled` attribute in an `<apama>` element and in the new `<universal-messaging>` element indicates whether the adapter uses UM. The default behavior is that an adapter does not use UM. To configure an adapter to use UM, set `enabled` to true in the `<universal-messaging>` element. Also, set `enabled` to false in the `<apama>` element, if there is one. This indicates that the configuration specified in the `<apama>` element should be ignored and the configuration specified in the `<universal-messaging>` element should be used.

When you are upgrading an application to use Apama 5.2 and you are configuring it to use UM keep in mind that events must be sent to named channels. You might need to add channel specifications to events. For adapters, you can use the `transportChannel`, `presetChannel`, and `defaultChannel` attributes to set channel names. If you do not specify values for any of these attributes, but you use Apama Studio to add UM support to your project then a default channel name will be in place for adapters that use UM. If you do not use Apama Studio then you must at least specify a value for the `defaultChannel` attribute in IAF adapter configurations. For details, see "Event file format" in *Deploying and Managing Apama Applications* and "Configuring adapters to use UM" in *Connecting Apama Applications to External Components*.

Apama 5.2 supports only the 9.7 release of Universal Messaging. The *Supported Platforms* document for version 5.2 in the Apama documentation lists supported releases for all Apama components.

Details for using UM in an Apama application are in *Connecting Apama Applications to External Components*.

New and changed EPL statements in 5.2

Apama 5.2 includes the following enhancements to EPL:

- The new `send...to` statement supersedes the `enqueue...to` and `emit...to` statements. The `send...to` statement sends the specified event to the specified channel.

  You can no longer use `send` as an identifier since it is now a keyword.

  The `enqueue...to` and `emit...to` statements will be deprecated in a future release. Use the `send...to` statement instead.

- The new `monitor.subscribe()` and `monitor.unsubscribe()` statements subscribe/unsubscribe a context to the specified channel. The monitor that contains the statement and any other monitors in the same context are subscribed/ unsubscribed. Note that this is a special use of the `monitor` keyword.

- The new type `com.apama.Channel` represents a channel in the correlator. See "New feature for addressing contexts with channels in 5.2" on page 55.

Enhancements to Apama adapters in 5.2

Apama's Web Services Client adapter and Correlator-Integrated adapter for JMS now provide interfaces for registering and using custom mapping extensions that use Java Unified Expression Language (EL) resolvers and methods. In previous releases, while Apama provided EL resolvers and EL methods for you to use in mapping expressions, you could not add your own.

For details, see "Using custom mapping extensions" in *Connecting Apama Applications to External Components*.

Changes to correlator utilities in 5.2

Apama 5.2 enhances and changes correlator utilities as follows:

- `engine_connect`

  The `engine_connect` utility has a new parallel mode (`-m parallel` or `--mode parallel`) that uses a connection per channel between correlators and delivers events to the target correlator on the channel the events were sent from in the source correlator. Previous behavior of `engine_connect` is preserved in legacy mode (`-m legacy` or `--mode legacy`), which uses one connection between two correlators and delivers all events to the default channel.

All connections to the correlator are now persistent. Consequently, the `engine_connect` utility no longer needs to provide the `--persistent` option and this option has been removed.

There is a change in behavior when disconnecting. Previously, when you specified the `-x` option without also specifying the `-c` option the two correlators remained connected even though the source correlator stopped sending events to the target correlator. In this release, specification of the `-x` option without also specifying the `-c` option disconnects the two correlators.

- `engine_debug`

  When you plan to run this utility you must specify the `-g` (or `--nooptimize`) option when you start the correlator. This option disables optimizations that hinder debugging. In previous releases, you could run `engine_debug` even if `-g` was not specified when the correlator was started. Apama Studio automatically uses the `-g` option when it starts a correlator from a debug launch configuration. However, if you connect Apama Studio to an externally started correlator and you want to debug in that correlator then you must ensure that `-g` was specified when the externally-started correlator was started.

- `engine_inspect`

  The new `-P` option displays the name of any plug-in receivers, what channels each plug-in receiver is subscribed to, and the number of entries on the input queue of each plug-in receiver. A plug-in receiver is a correlator plug-in that is subscribed to at least one channel.

  The new `-R` option displays the names of any external receivers, their addresses, what channels each external receiver is subscribed to, and the number of entries in the output queue of each external receiver.

  The `-x` option now also lists the channels each context is subscribed to.

- `engine_management`

  When you specify the `-xr` or `-xs` option, you must now specify the component ID as *`physical_ID`*/*`logical_ID`*.

- `engine_receive`

  The new `-C` or `--logChannels` option causes the `engine_receive` utility to include the channel on which an event is received in `engine_receive` output. If you then use `engine_receive` output as input to the `engine_send` utility the events will be delivered back to their channels.

- `engine_send`

  It is now possible to prefix an event with a quoted string that denotes the channel that event is to be sent on. The prefix string follows the same escaping rules as string in events. For example, an entry in a `.evt` file might be: `"MyChannel",Tick("SOW", 35).`

The new `-c` *`channel`* option identifies the delivery channel for events for which a delivery channel is not specified.

- `engine_watch`

  This utility now provides the following information:

`Events on input context queues`	The total number of events on the input queues of all public contexts in the correlator.
`Most backed up input queue`	The input queue that has the most events waiting to be processed.
`Most backed up queue size`	The number of events on the input queue that has the most events waiting to be processed.
`Slowest receiver`	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.

- `extract_replay_log`

  When you run this script on a correlator input log you can specify the `-c` or `--correlator` option. This option specifies that the script that `extract_replay_log` 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. While this option is not new in this release, it was previously undocumented.

  Support for replay logs has been removed. The `extract_replay_log` utility is now for extracting only input logs.

Windows compiler and .NET version updates in 5.2

Apama 5.2 has upgraded from Microsoft Visual Studio 2008 (Visual C++ v9) to Visual Studio 2013 Update 2 (Visual C++ v12). Consequently, you must rebuild the following components with the new compiler:

- C and C++ correlator plug-ins

- C and C++ IAF transport/codec plug-ins
- C and C++ code that uses the engine client API

For .NET clients, Apama now supports .NET 4.5.1 instead of 2.0. It is no longer possible to build .NET 2.0 or 3.5 clients with the Apama libraries.

In addition, you should consider the following:

- Visual Studio 2013 uses a different format for its C++ project files, which now have the `.vcxproj` extension rather than `.vcproj`. Visual Studio's migration wizard can usually convert the project files automatically. Alternatively, it is simple to create new projects in Visual Studio 2013 (from scratch or based on Apama samples) and re-add existing source files and project customization to them. Creating new projects might result in cleaner configuration files.
- The Apama C and C++ API samples all have new Visual Studio 2013 project files. Some non-standard preprocessor macro definitions that were present in older versions of the samples have been removed, for example, `_APBUILD_WIN32_ALL_` and `_AMD64_`. The samples now consistently use Visual Studio's standard macros instead (`_WIN32` and `_M_AMD64`). If you are incorporating existing code that depends on the old macros you might want to re-add the old macros to your project properties. To do this, in the Visual Studio, select your project and then select **C/C++** > **Preprocessor** > **Preprocessor Definitions**.
- When planning migration, consider that the C++ compiler in Visual Studio 2013 is considerably more modern than the 2008 version, and also less tolerant of ambiguous and potentially unsafe code patterns. It is likely that some time may be required to fix compiler errors after moving to the new version. The recommendation is to enable **'warnings as errors'** in your projects where possible. This encourages good coding patterns and catches errors earlier in the development process. In particular, it helps you catch errors that might cause subtle, difficult to diagnose errors.
- .NET code that uses the engine client API may need to be rebuilt. The `.csproj` project files will need to be upgraded if you are using the new compiler.

Linux compiler updates in 5.2

Apama 5.2 builds on Red Hat Enterprise Linux (RHEL) 6 Update 1 with GCC 4.4.5. Apama is certified for use on this platform.

You should rebuild the following components with the new compiler:

- C and C++ correlator plug-ins
- C and C++ IAF transport/codec plug-ins
- C and C++ code that uses the engine client API

If you are using SUSE Linux Enterprise Server (SLES) 11 Update 3 with the default shipped GCC 4.3 compiler, note that while Apama 5.2 is certified to work on this platform, you will see a **no version information available** warning when building plug-ins against Apama. This is because the compiler used for building Apama was GCC 4.4.5. At runtime, this should not present any problems.

Upgrading to Apama-supplied JDBC and ODBC database drivers in 5.2

As in past releases, Apama 5.2 provides JDBC and ODBC database drivers. For Apama 5.2, these drivers have been upgraded and configured slightly differently than in past releases.

The drivers provided in previous releases are incompatible with Apama 5.2. The drivers provided in Apama 5.2 are incompatible with previous Apama releases. If you have pre-Apama 5.2 projects that use the previously-supplied drivers and you want to run those projects with Apama 5.2 then you must manually convert those projects to use the Apama 5.2 drivers.

To upgrade your project to use Apama 5.2 ODBC drivers, change your application so that it uses an ODBC driver supplied with Apama 5.2. See "Registering your ODBC database DSN on Windows" in *Connecting Apama Applications to External Components*.

To upgrade your project to use Apama 5.2 JDBC drivers, change your application to use the new JDBC URLs and class names.

MSSQL

Old path:	*apama_install_dir*\lib\pgsqlserver.jar
New path:	*apama_install_dir*\lib\eysqlserver.jar
Old URL:	jdbc:progress:sqlserver://*HOSTNAME*::*PORT*;databaseName=*DATABASENAME*
New URL:	jdbc:sag:sqlserver://*HOSTNAME*::*PORT*;databaseName=*DATABASENAME*
Old class name:	com.prgs.jdbc.sqlserver.SQLServerDriver
New class name:	com.apama.jdbc.sqlserver.SQLServerDriver

Oracle

Old path:	*apama_install_dir*\lib\pgoracle.jar

New path:	*apama_install_dir*\lib\eyoracle.jar
Old URL:	jdbc:progress:oracle://*HOSTNAME*::*PORT*;databaseName=*DATABASENAME*
New URL:	jdbc:sag:oracle://*HOSTNAME*::*PORT*;databaseName=*DATABASENAME*
Old class name:	com.prgs.jdbc.oracle.OracleDriver
New class name:	com.apama.jdbc.oracle.OracleDriver

DB2

Old path:	*apama_install_dir*\lib\pgdb2.jar
New path:	*apama_install_dir*\lib\eydb2.jar
Old URL:	jdbc:progress:db2://*HOSTNAME*::*PORT*;databaseName=*DATABASENAME*
New URL:	jdbc:sag:db2://*HOSTNAME*::*PORT*;databaseName=*DATABASENAME*
Old class name:	com.prgs.jdbc.db2.DB2Driver
New class name:	com.apama.jdbc.db2.DB2Driver

Removed and deprecated features in 5.2

The following features are removed or deprecated in this release:

- Support for the correlator replay log has been removed. You can now replay correlator behavior only with a correlator input log.

  The following options were needed when starting a correlator that would capture or run a replay log. These options are no longer needed and they have been removed:

  `--replayLog`

  `--replayMode`

  With the removal of the replay log, the `extract_replay_log` utility now extracts data from input logs only.

When using Apama Studio you can specify an input log but not a replay log in the **Correlator Arguments** tab of the **Correlator Configuration** dialog. The **Replay log mode** field has been removed. from the **Correlator Arguments** tab.

Replay log support has also been removed from Apama Studio Ant features. Trying to run an Ant deployment script with a replay log will result in an error when trying to start the correlator.

- The following correlator start-up options have been removed:

  `--highThroughput`

  `--lowLatency`

  When using EMM, you can no longer select these options in the correlator **Configurations** tab.

  These options were useful when running Apama on single-core machines. With the recommendation to always run Apama on multiple core machines, these options are no longer needed.

- Now that all connections to the correlator are persistent, the `engine_connect --persistent` option is no longer needed and it has been removed.

- The following methods in the MemoryStore `RowValue` API class have been deprecated and will be removed in a future release:

```
public final void setString(int typedIndex, int maxSizeHint, String v)
public final void setString(int typedIndex, int maxSizeHint,
   byte[] utf8ByteString)
public final void setInteger(int typedIndex, int maxSizeHint, long v)
public final void setFloat(int typedIndex, int maxSizeHint, double v)
public final void setBoolean(int typedIndex, int maxSizeHint, boolean v)
```

  In place of these methods, use the `set*()` methods without the `maxSizeHint` argument.

- The Payload Extraction correlator plug-in has been deprecated as of Apama 5.0. While it has not yet been removed because of internal requirements, it will be removed in a future release and its use is strongly discouraged.

  The dictionary-format payload is considerably more efficient than the string-format payload in almost all cases. Third-party adapters should now be using dictionary-format payloads.

- The following Event Modeler functions are deprecated and will be removed in a future release:

  `ADD_EXTRAPARAM`

  `GET_EXTRAPARAM`

  `HAS_EXTRAPARAM`

  In place of these deprecated functions, use the following functions:

  `DICT_GET`

`DICT_GETORDEFAULT`

`DICT_HASKEY`

`DICT_SET`

- The following overloading of the C++ plug-in API method for sending an event has been deprecated:

  ```
  sendEventTo(const char* event, AP_uint64 targetContext,
     AP_uint64 sourceContext) = 0;
  ```

  It will be removed in a future release. In place of that overloading, use the following method:

  ```
  sendEventTo(const char *event, AP_uint64 targetContext,
     const AP_Context &source) = 0;
  ```

- With the addition of the `send...to` statement, `send` is now a keyword and you can no longer use it as an identifier unless you escape it with a hash symbol, for example, `#send`.

While the following EPL statements are not deprecated in this release, they will be deprecated in a future release. Use the EPL `send...to` statement instead.

- `emit` and `emit...to`
- `enqueue...to`

Miscellaneous enhancements and changes in 5.2

Apama 5.2 includes the following enhancements:

- In an IAF adapter configuration file, the new default behavior is that all sinks defined for an Apama event correlator configuration now receive the events generated by the IAF Semantic Mapper. To change the default behavior, add the `sendEvents="false"` attribute to each `<sink>` element that represents a component that should not receive events. In previous releases, the default behavior was that only the first sink would receive events.
- Several changes have been made to the `EngineClient`, `EventService` and `ScenarioService` layers in Java and .NET to make it easier to use event and property change listeners safely without risking thread deadlock situations. As a result of these changes, Apama client developers might notice that engine client property change listeners now fire asynchronously (on a separate dedicated thread), which is a slight change in behavior. See the release notes for details.
- When using EDA events in Apama applications, non-EDA namespaces are now allowed. Also, namespaces that contain colons (`:`) can now be converted to Apama package names.
- In Dashboard Builder, when you define the **Send event** command, you can optionally specify the channel on which to send the event.

- The new **Correlator Time Format** dashboard function lets you convert a correlator timestamp to epoch time or to a date/time format you specify.
- The BigMemory MemoryStore driver now sets the Java `type` of each search attribute, which makes it easier to use the search attributes from other products. For example, Software AG's Presto can extract the `ehcache.xml` configuration from a running BigMemory instance.
- The distributed MemoryStore `RowValue` API class has a number of usability improvements in its API, including easier to use set and get methods. (The old-style set methods have been deprecated.) There is also a new `RowValueHelper` class that allows more efficient allocation of `RowValue` objects, and simplifies getting, setting and iterating over the contents of a `RowValue` object.
- Apama 5.2 incorporates ICU (International Components for Unicode) Timezone Data update 2014e, which is the most recent update at the time of release. This will update timezone data used by the correlator and TimeFormat correlator plug-in.
- C++ correlator plug-ins can be declared as nonblocking to prevent creation of unneeded processing threads. If a method in a nonblocking plug-in might block you can override the nonblocking designation for that method.
- The C++ plug-in API method for sending an event has a new overloading, which you should now use:

  ```
  sendEventTo(const char *event, AP_uint64 targetContext,
    const AP_Context &source) = 0;
  ```

  The following is deprecated and will be removed in a future release:

  ```
  sendEventTo(const char* event, AP_uint64 targetContext,
    AP_uint64 sourceContext) = 0;
  ```

6 What's New in Apama 5.1.1

Apama 5.1.1 includes the following enhancements:

- The correlator now parses incoming events from different connections in parallel. This improves performance when many adapters or clients are sending events to the correlator.
- Apama dashboards now support IBM WebSphere Application Server 8.5.5. Apama's correlator-integrated messaging for JMS feature also now supports WAS 8.5.5.
- Apama Studio can now export an Ant launch configuration that accommodates changes to your application. This means you need to export the launch configuration only once even if you subsequently add, remove or edit a file in your application. In previous releases, to ensure that your exported scripts were in sync with your application you had to re-export the launch configuration each time you changed your application.

  To turn on the new export-once behavior, select the new **Generate initialization list during launch** option when exporting an Ant launch configuration from Apama Studio. The exported launch configuration contains the location of your project directory. When the exported deployment script is executed, the file initialization list is generated from the project directory.

  If you select **Generate initialization list during launch** then any scenario files are always converted to monitor files at the time of injection to the correlator.

  For details, see *Exporting a Launch Configuration* in *Using the Apama Studio Development Environment* (renamed to *Using Apama with Software AG Designer* in version 9.9).
- Search attributes can now be exposed for Apama MemoryStore application tables that are stored in Terracotta BigMemory caches. Turning this configurable capability on enables BigMemory clients to query data stored by Apama applications in BigMemory caches. The default is that search attributes are not exposed.

  For more information, see *Using the distributed MemoryStore* in *Developing Apama Applications*.
- Apama applications now support the use of Software AG Event-Driven Architecture (EDA) event types. In Apama Studio, you can generate Apama event type definitions from EDA event definition schemas and you can map these Apama event definitions to JMS messages that communicate with EDA applications. For details, see *Using EDA Events in Apama Applications* in *Deploying and Managing Apama Applications*.

7 What's New in Apama 5.1

What's New in Apama 5.1 summarizes the new, enhanced, and changed features in Apama 5.1.

If you are upgrading from a version of Apama prior to 5.0 then consult the appropriate What's New and Migration information available with the intermediate versions. For further assistance, contact customer support.

New distributed MemoryStore

The Apama MemoryStore now provides the ability to create distributed stores in which data can be shared by applications running in multiple correlators. Distributed stores are supported by distributed caching software from a variety of third-party vendors. Apama provides a driver for integrating the distributed MemoryStore with the Terracotta BigMemory Max distributed caching software. Apama also provides a Service Provider Interface (SPI) for creating drivers to use with other third-party distributed cache providers.

In Apama Studio, you can configure a correlator to use the distributed MemoryStore by selecting **Distributed MemoryStore Support** in the **Correlator Configuration** dialog. See "Correlator arguments" in *Using the Apama Studio Development Environment* (renamed to *Using Apama with Software AG Designer* in version 9.9).

For information on using the distributed MemoryStore and for creating drivers for third-party distributed caching software, see "Using the MemoryStore" in *Developing Apama Applications*.

New EPL exception handling mechanism

EPL now supports the try-catch exception handling structure for processing runtime errors. The try-catch statement's BNF definition is:

```
tryCatchStatement ::= try block1 catch(Exception variable) block2
```

The statements in each block must be enclosed in curly braces. For example:

```
using com.apama.exceptions.Exception;
...
action getExchangeRate(
   dictionary<string, string> prices, string fxPair) returns float {
   try {
      return float.parse(prices[fxPair]);
   } catch(Exception e) {
      return 1.0;
   }
}
```

An exception that occurs in `try block1` causes execution of `catch block2`. Two new types have been added to support exception handling:

- `com.apama.exceptions.Exception` — A variable of this type contains an exception message and an exception type. It also contains a sequence of `com.apama.exceptions.StackTrackElement` objects. The sequence represents the stack trace for when the exception was first thrown. `Exception` objects have methods for accessing the exception message, the exception type, and the stack trace.

- `com.apama.exceptions.StackTraceElement` — A variable of this type contains information for one stack trace entry. A `StackTraceElement` object has methods for accessing the details for the stack trace entry it represents.

`try`, `catch` and `throw` are now keywords. If you want to use one of these words as an identifier you must prefix it with a hash symbol (`#`), otherwise it is an error.

For details, see *Developing Apama Applications in EPL*, *Defining What Happens When Matching Events are Found*, *Catching exceptions*.

New support for managing and monitoring with REST APIs

You can now monitor Apama components with the Apama REpresentational State Transfer (REST) HTTP API. This provides monitoring capabilities to third-party managing and monitoring tools or to any application that supports sending and receiving XML documents over the HTTP protocol.

Apama components expose several URIs which can be used to either monitor or manage different parts of the system. Generic management URIs are exposed by most Apama components, while other URIs are exposed only by specific types of components. 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. For example, to set a component's log level, the `/logLevel` URI accepts an HTTP `PUT` request containing an XML document that specifies the log level.

For example: `GET http://localhost:15903/correlator/status` returns an XML document that contains:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="/resources/transform.xslt"?>
<map name="apama-response">
   <prop name="numConsumers">0</prop>
   <prop name="numOutEventsQueued">0</prop>
   <prop name="numOutEventsUnAcked">0</prop>
   <prop name="numOutEventsSent">0</prop>
   <prop name="uptime">67657</prop>
   <prop name="numMonitors">0</prop>
   <prop name="numProcesses">0</prop>
   <prop name="numJavaApplications">0</prop>
   <prop name="numListeners">0</prop>
   <prop name="numEventTypes">0</prop>
   <prop name="numQueuedFastTrack">0</prop>
   <prop name="numQueuedInput">0</prop>
   <prop name="numReceived">0</prop>
   <prop name="numFastTracked">0</prop>
   <prop name="numEmits">0</prop>
   <prop name="numProcessed">0</prop>
   <prop name="numSubListeners">0</prop>
   <prop name="numContexts">1</prop>
   <prop name="virtualMemorySize">262540</prop>
   <prop name="numSnapshots">0</prop>
   <prop name="numInputQueuedInput">0</prop>
   <prop name="mostBackedUpInputContext"><none></prop>
   <prop name="mostBackedUpICQueueSize">0</prop>
   <prop name="mostBackedUpICLatency">0.0</prop>
</map>
```

For more information on using the REST API, see "Managing and Monitoring over REST" in *Managing and Monitoring Apama Applications*.

New support for creating correlator plug-ins using Java

Custom correlator plug-ins can now be written in Java as well as in C and C++. When a Java plug-in is injected to the correlator, it is analyzed and any suitable method in the plug-in can be called from applications running in the correlator. Correlator plug-ins in Java are deployed using JMon applications and are packaged in a `.jar` file.

Correlator plug-ins that are written in Java can be created by using Apama Studio or from scratch.

- To use Apama Studio, see "Adding an EPL Plug-in written in Java" in *Using the Apama Studio Development Environment* (renamed to *Using Apama with Software AG Designer* in version 9.9).
- To write a correlator plug-in in Java from scratch, see "Developing Correlator Plug-ins in Java" in *Developing Apama Applications*.

New headless mode generation of ApamaDoc

On Windows platforms, you can now generate ApamaDoc in headless mode. As a standalone operation from the command line, run the `apamadoc.bat` script, which is in the `%APAMA_HOME%\bin` folder. The `apamadoc.bat` file uses the `%APAMA_HOME%\utilities\apamadoc.xml` ant script to generate ApamaDoc.

For more information, see *Developing Apama Applications*, *Generating Documentation for Your EPL Code*, *Generating ApamaDoc in headless mode*.

New support for PySys test framework

PySys, an open source automated system testing framework, is recommended as the best way to test Apama applications developed with EPL, Event Modeler, or JMon. The open source PySys project operates on generic elements such as starting and stopping processes, and searching text files for correct or incorrect output. In addition, Apama includes extensions to PySys that let you test correlators, IAF components, EPL injection, sending and receiving events, and similar operations. The PySys framework can start adapters, which enables you to perform end-to-end testing of an entire system.

Samples for how to use PySys are in the `samples\pysys` folder of your Apama installation. Reference documentation for PySys and Apama extensions is in the `doc\pydoc` directory of your Apama installation.

Dashboard login redirect file name has changed

The `loginRedirect.html` file has been renamed to `loginRedirect.jsp`.

In previous releases, the `loginRedirect.html` file appeared if you tried to display a dashboard before you logged in or after your session expired. With this release, the new name of this file is `loginRedirect.jsp`.

The default behavior of `loginRedirect.jsp` is to redirect the user to the dashboard's form authentication page. If you have implemented a custom login, you must create a custom version of `loginRedirect.jsp`. For example:

```
<meta HTTP-EQUIV="REFRESH" content="0; url="../userLogin">
<html>
   <head>
      <title>Apama: Login Redirect from Dashboard</title>
      <meta http-equiv="pragma" content="no-cache">
      <meta http-equiv="cache-control" content="no-cache">
      <meta http-equiv="expires" content="0">
   </head>
   <body/>
</html>
```

In this sample custom `loginRedirect.jsp` file, `"../userLogin"` is a custom login page.

To enable your custom `loginRedirect.jsp` file, replace the provided version of it in the following locations:

- *APAMA_INSTALL*`\etc\template.war`
- *APAMA_INSTALL*`\etc\dispTemplate.war`

After your custom `loginRedirect.jsp` file is in place, you must re-deploy your dashboards.

New EPL keywords

New EPL keywords are listed below. If you want to use one of these words as an identifier you must prefix it with a hash symbol (`#`), otherwise it is an error.

- `catch`
- `optional`
- `static`
- `throw`
- `try`

Miscellaneous enhancements in 5.1

Apama 5.1 includes the following enhancements:

- Correlator Deployment Packages (CDPs) can now contain `.jar` files.
- In Apama Studio, you can now export a project's Web Services Client adapter configuration to an archive file. You can then import the archive file into any Apama project.
- The Apama correlator-integrated adapter for JMS now supports Software AG's Universal Messaging 9.5.2.
- In previous releases, when using the Apama correlator-integrated messaging adapter for JMS, you had to explicitly obtain and install the `javax.jms.jar` file. This file is now provided with Apama and installed as part of Apama installation. Manual steps are no longer required.
- In the **Event Mappings** tab for the correlator-integrated messaging adapter for JMS and for the Web Services Client adapter, Apama Studio now displays error markers if events that were previously mapped are missing. You must correct the event mappings before you can run the adapter.
- The `com.apama.jmon.Unloadable` interface has been added to JMon. For any monitor that implements this interface, the `Unloadable.onUnload()` method will be called when the application is being unloaded, for example, due to an `engine_delete` request.
- Dashboards now support the following functions. Details for using these functions are in the *Dashboard Function Reference*.
  - **Init Local Variable** initializes a local variable to a specified value.
  - **Quick Set Sub** sets a substitution string to a specified value.
- For dashboard processes, the `filterInstance` attribute in the `apama-macros.xml` file now defaults to true. This ensures that users can see only the instances that are owned by them. To change this behavior, edit the `apama-macros.xml` file or create the DataView instances with `"*"` as the owner.
- As of Apama 5.0, Apama applications running in the correlator can make use of Apama *out of band notifications*. Out of band notifications are events that are automatically sent to all public contexts in a correlator whenever any component (an IAF adapter, dashboard, another correlator, or a client built using the Apama SDKs) connects or disconnects from the correlator. (This was not previously listed in the *Release Notes*.)

Support removed from Apama 5.1

Support for the following features has been removed from Apama:

- IAF-based adapter for JMS
- Routers - A router is a specialized correlator that is optimized to partition events so they go to different correlators. With this release, any correlator can be used as a router and the documentation includes information and instructions for configuring this.
- The engine client simple/bean receiver in the engine client API for Java and for `.NET`. Applications still using this deprecated event receiving mechanism (that is, using the `setReceiveEnabled()` Java method or `ReceiveEnabled .NET` property) should be changed to use the `addConsumer()` method instead, which will require moving event handling code out of the property changed listener into a new `IEventListener` implementation.
- Third-party products
    - Actional adapter
    - Control Tower
    - Corticon
    - OpenEdge
    - Savvion adapter
    - Sonic ESB service

Migrating Apama applications

For up to date information about migrating to Apama 5.1, refer to the Apama Knowledge Base articles on Empower (empower.softwareag.com).

8 What's New in Apama 5.0.1

What's New in Apama 5.0.1 summarizes the new, enhanced, and changed features in Apama 5.0.1.

If you are upgrading from a version of Apama prior to 5.0 then consult the appropriate What's New and Migration information available with the intermediate versions. For further assistance, contact customer support.

Linux runtime performance enhancement

On Linux 64-bit systems, you can specify whether you want the correlator to use the compiled runtime or the interpreted runtime. The compiled runtime is new in Apama 5.0.1.

When you start the correlator, specify `--runtime compiled` to use the new compiled runtime feature. The interpreted runtime is invoked by default or you can specify `--runtime interpreted`.

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 managing listeners and searching for matches among listeners typically show a smaller performance improvement.

Other than performance, the behavior of the two runtimes is the same except

- 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.
- 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.

Changes to the Apama Database Connector

LogCommands and LoqQueries

Two new properties, `LogCommands` and `LoqQueries` are introduced in this release. These properties specify whether or not the start and completion of commands and queries are written to the IAF log file. A value of "`true`" (the default) logs this information; a value of "`false`" turns logging off. This is useful in cases where logging the start and completion of a high rate of commands (many hundreds or thousands per second) does not add usable information. The value of these properties is specified in the **XML Source** tab of Apama Studio's ADBC adapter editor. For more information, see "Configuring an ADBC adapter" in *Connecting Apama Applications to External Components*.

Integrating JMon applications with Corticon

Apama 5.0.1 is now more tightly integrated with the Progress® Corticon® Business Rules Management System. New features in this release make it convenient to invoke a Corticon Decision Service from a JMon application. A new wizard in Studio allows you to automatically generate the JMon events that represent a Corticon vocabulary in an Apama application along with the monitors that listen for these events.

You can also automatically generate the JMon events that let you configure the Corticon Decision Service. In addition Apama Studio will automatically set up Apama projects and adds references to the required Corticon jar files.

For more information on using this feature to invoke a Corticon Decision Service from a JMon application, see *Integrating JMon Applications with Corticon*.

Correlator-integrated adapter for JMS bundle name changes

Apama recommends that you use the correlator-integrated adapter for JMS wherever possible instead of the legacy Apama adapter for JMS.

When opened in Apama 5.0.1, existing Apama projects that use the adapters (with the old bundle names) will display the new names.

Improved profiling analysis

Starting with Apama Release 5.0.1 you can use profiling data that has been previously stored on disk in a `.csv` file to create a snapshot for profiling analysis in Apama Studio. For example, you can capture a profile on a testing system (which may be running on an operating system other than Windows) and transfer it to a workstation running Apama Studio for analysis. This data would typically be collected using the `engine_management` tool.

Only `.csv` files that have been generated using Apama Release 5.0.1 or later can be used to create the snapshot.

For more information on this feature, see "Profiling EPL Applications" in *Using the Apama Studio Development Environment* (renamed to *Using Apama with Software AG Designer* in version 9.9).

For more information on the `engine_management` utility, see "Shutting down and managing components" in *Deploying and Managing Apama Applications*.

5.0 features now documented

Documentation has been added for the following Apama 5.0 features:

- Dashboard tree controls, which let you create rich and compact visual presentations of hierarchical data. A tree control is most often used in a multi-panel application for display navigation. A tree control can also be used in any application where hierarchical data is most effectively displayed using expandable/collapsible tree nodes.

  For additional details, see *Building and Using Dashboards*, *Reusing Dashboard Components*, *Working with multiple display panels*, *Using tree controls in panel displays*.

- `TRACE` log level, which generates the most verbose output. You can specify `TRACE` anywhere you can specify the log levels provided in earlier releases: `OFF`, `CRIT`, `FATAL`, `ERROR`, `WARN`, `INFO`, `DEBUG` and it applies to the correlator, router, IAF and sentinel agent.

  For additional details, see *Deploying and Managing Apama Applications*, *Correlator Utilities Reference*, *Shutting down and managing components*, *Setting logging attributes for packages, monitors and events*.

- New fields in the status lines the correlator logs periodically. The `lcn`, `lcq`, and `lct` fields provide metrics about the most backed up queue, and the `runq` field indicates the number of contexts that have work to do but are not currently running.

  For additional details, see *Deploying and Managing Apama Applications*, *Correlator Utilities Reference*, *Starting the event correlator*, *Logging correlator status*.

- The EPL utility event that provides access to the correlator Management interface has additional actions that return information about the correlator that the Management interface is running in. These actions are defined in the `com.apama.correlator.Component` event. There are `engine_management` utility options that provide similar behavior. The correlator logs all of these values in its log file at start up. The available actions are:

  - `getHostname()` - Returns the host name of the host the correlator is running on. The host name is dependent on the environment's name resolution configuration, and the name can be used only if the name resolution is correctly configured. The name is the same as that logged in the correlator log file, for example, `dev3.acme.com`.
  - `getComponentPort()` - Returns the port the correlator is running on.
  - `getComponentPhysicalId()` - Returns the physical ID of the correlator.
  - `getComponentLogicalId()` - Returns the logical ID of the correlator.
  - `getComponentName()` - Returns the name that is used to identify the correlator. You can set this name by specifying the `-N` correlator command line flag (or by

means of the `extraArgs` attribute in the Ant macros). The default name of the correlator is `correlator`.

To use the Management interface, add the `Correlator Management` bundle to your Apama project. Alternatively, you can directly use the EPL interfaces provided in `APAMA_HOME\monitors\Management.mon`.

For additional details, see *Developing Apama Applications*, *Using Correlator Plug-ins in EPL, Using the Management interface*.

Specifying Dashboard Builder options in Apama Studio

You can now specify options to be used when an Apama project opens Dashboard Builder. You can use this feature to override the default values used when starting the Dashboard Builder. The options correspond to the command line arguments available when you manually start the Dashboard Builder executable. For example, you can use a command line option to change the default name used by the correlator to identify the Dashboard Builder component. By default, the correlator logs messages such as:

```
2013-03-22 07:50:27.355 INFO  [9904] - Sender Dashboard Builder:
  achan (0000000001AFEA30)
  (component ID 10514461075117768704/10514461075117768704)
  connected from 127.0.0.1:52057
```

If you use the command line `--name My_Builder` the correlator logs messages as:

```
2013-03-22 07:51:07.272 INFO  [9904] - Sender My_Builder (0000000007469010)
  (component ID 10515944982030123008/10515944982030123008)
  connected from 127.0.0.1:52088
```

You specify these options on the **Dashboard Builder Options** tab, available when you display a project's Properties page in Apama Studio and select **Apama > Dashboard Properties**.

For details, see "Specifying Dashboard Builder options" in *Building Dashboard Clients*.

9 What's New in Apama 5.0

What's New in Apama 5.0 summarizes the new, enhanced, and changed features in Apama 5.0.

For information about upgrading Apama applications, see "Migrating from Apama 4.3 " on page 113.

Changes to EPL features in Apama 5.0

Apama 5.0 provides many enhancements to EPL. These enhancements are described in two sections in order to highlight the enhancements that might require changes to applications that are being migrated from Apama 4.3. Be sure to read both topics to learn about all Apama 5.0 EPL enhancements and changes.

"EPL changes without migration implications" on page 88

"EPL changes that might have migration implications" on page 91

EPL changes without migration implications

This topic describes EPL enhancements. If you use these new EPL features in applications that you are migrating from Apama 4.3 you do not need to change the application other than to implement these new features.

See also: "EPL changes that might have migration implications" on page 91.

Addition of decimal primitive type

EPL now supports the `decimal` primitive type. When perfect accuracy in working with decimal fractions is a requirement, use the `decimal` type in place of the `float` type. When you are not working with decimal values then using `float` types is the right choice. For example, it makes sense to use `decimal` types to compute 3.9% interest on $127,729.23 if it is important that the answer be exact. Conversely, there is no reason to use a `decimal` type to determine the area of a circle, for example, because pi is not a decimal fraction. When extremely small floating point variations are acceptable, you might choose to use the `float` type to obtain better performance.

A `decimal` type is a signed, decimal, floating point number. Either a decimal point (.) or an exponent symbol (e) must be present within the number for it to be a valid decimal. Also, there must be a decimal suffix (`d`) to distinguish it from a `float` type.

Streams of `decimal` type items are supported. The built-in aggregate functions that you can specify in stream query projections support `decimal` types.

Correlator plug-ins, Event Modeler, DataViews, and dashboards do not support use of the `decimal` type.

New string type methods

Apama 5.0 provides the following new `string` methods:

- The new `string.intern()` method is an optimization that can reduce memory use. This method takes no arguments, returns the interned version of the string that it is called on, and marks the string as being interned, which means that the correlator uses the same string object for every subsequent occasion that the same string is parsed.
- The new `string.replaceAll(`*`string1, string2`*`)` method makes a copy of the string that the method is called on, replaces instances of *`string1`* in the copy with instances of *`string2`*, and returns the updated copy.
- The new `string.split(`*`string`*`)` method splits the string that is the argument at occurences of the string that the method is called on and returns a sequence that contains the new strings.
- The new `string.tokenize(`*`string`*`)` method divides the `string` argument into tokens where each token is separated from its neighbors by one or more delimiters in the string that the method is called on. The `tokenize()` method returns a sequence of strings.

Allowable key types in dictionaries

It is now possible to use any comparable type as the key in a dictionary. The comparable types are:

- `boolean`
- `decimal`
- `float`
- `integer`
- `string`
- `context`
- `dictionary` if it contains items that are a comparable type
- `event` if it contains only comparable types
- `location`
- `sequence` if it contains items that are a comparable type

New dictionary methods

Apama 5.0 provides the following new `dictionary` methods:

- The `dictionary.getOr(`*`key, alternative`*`)` method either retrieves an existing item by its key or returns the specified alternative.
- The `dictionary.getOrDefault(`*`key`*`)` method either retrieves an existing item by its key or returns a default instance of the dictionary's item type if the dictionary does not contain the specified key.

- The `getOrAdd(`*`key`*`, `*`alternative`*`)` method either retrieves an existing item by its key or if the dictionary does not already contain an item with the specified key then it adds the specified key to the dictionary with the specified value and then returns that value.
- The `getOrAddDefault(`*`key`*`)` method either retrieves an existing item by its key or if the dictionary does not already contain an item with the specified key then it adds the specified key to the dictionary with a default-constructed value and returns that value.

The `getOr()` methods let you avoid calls to the `hasKey()` method before you look up a key. This leads to safer, simpler, and faster code.

Before Apama 5.0, the `dictionary.getOr()` method was called `dictionary.getDefault()`. The `getDefault()` method remains only for backwards compatibility and should not be used. It is deprecated and will be removed in a future release. Use the `dictionary.getOr()` method instead.

New sequence method

The new `sequence.setCapacity(`*`integer`*`)` method sets the amount of memory initially allocated for the sequence. Note that this does not limit the amount of memory the sequence can use. By default, as you add more elements to a sequence, the correlator allocates more memory. Calling `sequence.setCapacity()` can improve performance because it removes the need to add more memory each time you add an element to the sequence. For example, consider a sequence that will contain at least 1000 elements. A call to `setCapacity(1000)` removes the need to allocate additional memory unless more than 1000 elements are added. A call to this method does not change the behavior of your code.

New context method

The new `context.isPublic()` method returns a Boolean value that indicates whether the context is public.

New chunk methods

Apama 5.0 provides the following new `chunk` methods:

- `empty()` – returns true if the chunk is empty. This lets you distinguish between a chunk that contains a default initialization value and a chunk that has been populated by a correlator plug-in.
- `getOwner()` – returns a string that contains the name of the correlator plug-in that the chunk belongs to. This method returns an empty string if the chunk is empty.

Enhancement to MemoryStore plug-in

When you expose a MemoryStore table as a DataView you can now specify the display name and/or the description for the exposed DataView. Previously, the MemoryStore always used a default display name and description.

Names no longer reserved for future use

The names `open`, `close` and `reset` are no longer reserved for future use. A custom aggregate function can have members with these names.

EPL samples removed

The following sample event interfaces and monitors, some of which are specific to capital market solutions, have been removed from Apama 5.0:

- `DatabaseSupport.mon`
- `MultiLegOrderManagerSupport.mon`
- `OrderManagerSupport.mon`
- `ScenarioOrderService.mon`
- `SimpleExchangeSimulator.mon`
- `SimpleNonPersistentDatabase.mon`
- `SimplePriceGenerator.mon`
- `TickManagerSupport.mon`

EPL changes that might have migration implications

This topic describes EPL changes and enhancements. If you use these EPL features in applications that you are migrating from Apama 4.3 you might need to change the application before you can run it with the newer Apama release.

See also: "EPL changes without migration implications" on page 88, which describes EPL enhancements that do not require updates for migrating applications from Apama 4.3, other than to implement the new features.

Streams: prohibition of taking the rstream of an aggregate projection

In previous releases, expressions such as the following have technically been legal. In Apama 5.0, the use of the `rstream` keyword in conjunction with an aggregate projection has been withdrawn.

```
for a in aF within 10.0 select rstream sum(a.x)
```

Beginning with Apama 5.0. the taxonomy is that a projection can be:

- istream projection (the default)
- rstream projection (if `rstream` is specified)
- aggregate projection (if `having` and/or `select` expressions use an aggregate)

Additional changes in the EPL streams feature are:

- A `stream` type can now contain any type of item. Previously, a `stream` type was restricted to containing `boolean`, `float`, `integer`, `string`, or `event` type items.

- You can now specify a `having` clause to filter the items produced from the stream query's projection.

"persistent" is now an EPL keyword

As of Apama 5.0 it is no longer permissible to use `persistent` as an identifier, such as the name of a variable, monitor, or event. In this release `persistent` is a keyword used to designate a monitor as being persistent. Using it as an identifier will generate an error, unless you prefix it with a hash symbol (`#`).

The "call" statement is no longer valid

The use of the `call` statement has been deprecated for several releases and in Apama 5.0 it has been removed. To invoke an action, see "Defining actions" in *Developing Apama Applications with EPL*.

"streamsource" is reserved as a keyword

As of Apama 5.0, `streamsource` is now a keyword that is reserved for future use. Using it as an identifier will generate an error, unless you prefix it with a hash symbol (`#`).

"generates" is no longer a keyword

With this release, `generates` is no longer a keyword and it can be used as an identifier.

enqueue to default-constructed context

In previous releases, enqueuing to a default-constructed context rather than to an actual context silently did nothing. Beginning with Apama 5.0, code such as the following example will cause the correlator to terminate the monitor instance, which will assist developers in detecting problems caused by subtle coding bugs such as this.

```
monitor m {
   context c;
   action onload()
   {
      enqueue A() to c;
   }
}
```

If you enqueue an event to a sequence of contexts and one of the contexts has the default value then the correlator terminates the monitor instance.

Apama macros generate-scenario(s) - force is removed

The `force` parameter has been removed from the Apama macros for Ant (*`install_dir`*`\etc\apama-macros.xml`)that are used when re-building Apama projects. This means Apama Studio always regenerates the EPL code for a scenario in order to ensure that the project uses the correct, up-to-date code.

Changes to the correlator plug-in API

- The API version number has been stepped.

- EPL plug-in support for the string type has changed. Strings are now immutable and cannot be modified by the plug-in once created.
- Serialization and deserialization support has been removed. This means correlator plug-ins that support serialization will no longer compile until the obsolete code is removed.
- The `AP_Type` references obtained from the arguments (`AP_TypeList`) are now `const`.
- `AP_Chunk` can now be default-constructed. In previous versions, it had to be passed `const AP_Context &`.
- `ownerID` has been removed from `AP_Chunk`.
- The `pluginID()`, `registerChunk()`, and `deregisterChunk()` methods have been withdrawn from the `AP_Context` interface.
- The `AP_CorrelatorInterface` cannot be used from chunk destructors.
- The plug-in is responsible for deleting the old `AP_Chunk` when setting a new value for a chunk. You should use `AP_Type::chunkValue(AP_Chunk *)` to return the old `AP_Chunk *`, which should then be deleted. This behavior is new in this release.
- Various values are now `size_t` not `uint32`. Sequences larger than 2^^{32} elements can be manipulated on 64-bit platforms.
- Tweaked the `const`-correctness of various `AP_Type` methods.
- One plug-in can no longer use chunks produced by another.

Time Format plug-in

In previous releases, the Time Format parse functions returned `-1` when the specified string could not be parsed. Beginning with Apama 5.0, these functions return `NaN` on an error because in some circumstances `-1` is a legitimate result when there has been no error. This behavior applies to the following functions:

```
parse()
parseTime()
parseUTC()
parseTimeUTC()
parseWithTimeZone()
parseTimeWithTimeZone()
```

Changes to special actions

In Apama 5.0 the behavior of special actions has been made more uniform. Specifically:

- Beginning with this release, the `onBeginRecovery` and `onConcludeRecovery` actions must now contain no arguments and must return nothing. The correlator ensures this behavior and it is a compile-time error if you define one of these actions with any other signature
- No member or monitor-scoped variable may have the name of a special action. This applies to the following:

- Inside monitors these actions are `onload`, `ondie`, `onunload`, `onBeginRecovery`, `onConcludeRecovery`.
- Inside events these actions are `onBeginRecovery` and `onConcludeRecovery`.
- Inside aggregates these actions are `init`, `add`, `remove`, `value`, `onBeginRecovery`, and `onConcludeRecovery`.

parse() removed from some EPL types

EPL code that calls `parse()` or `canParse()` on the following types will now give an error at injection time:

- `action`
- `listener`
- `chunk`
- `stream`
- An event that contains an unparseable field
- A `sequence` that contains unparseable items
- A `dictionary` whose key or item type is unparseable

This is part of a simplification of the EPL type system, and means that incorrect code that would previously have injected but caused errors at runtime is now prevented from injecting to make the error more obvious.

indexOf() and sort() removed from some kinds of EPL sequence

EPL code that calls `indexOf()` or `sort()` (that is, actions requiring comparability of `sequence` elements) on sequences with the following element types will now give an error at injection time:

- `listener`
- `chunk`
- `stream`
- `action`
- nested sequences, dictionaries or events that are potentially cyclic (that is, those that have circular type references)

This is part of a simplification of the EPL type system, and means that incorrect code that would previously have injected but caused errors at runtime - or subtle logic bugs - is now prevented from injecting, to make the error more obvious.

Withdrawal of support for non-threadsafe plug-ins

The threadsafe capability flag for plug-ins has been removed, and all plug-ins must now be threadsafe. Customers who need to perform non-threadsafe operations will need create their own mutex within the plug-in itself.

Deprecation of dictionary.getDefault

The `dictionary.getDefault()` method has been deprecated. Use the new `dictionary.getOr()` method instead. See "New dictionary methods" in "EPL changes without migration implications" on page 88.

Deprecation of Payload Extraction plug-in

The Payload Extraction correlator plug-in is deprecated. All Apama adapter products now use dictionary-format payloads rather than string-format payloads. The dictionary-format payload is considerably more efficient in almost all cases. While string-format payloads are still accepted in Apama 5.0, this support, including the Payload Extraction correlator plug-in, is now deprecated and will be removed in a future release of the Apama platform. It is recommended that all third-party adapters are migrated to dictionary-format payloads as soon as possible.

The com.apama namespace is restricted

You cannot create EPL structures in the `com.apama` namespace. This namespace is reserved for the use of future Apama features.

New Apama client API support for decimal type

With release 5.0 the Apama Client Software Developement Kits for Java and `.NET` now have classes to represent the decimal field type.

Apama Client SDK for Java

A new class, `DecimalFieldValue` has been added to represent the Decimal Field Type. This will always represent the decimal value within a `BigDecimal` where possible, and is retrievable with calls to `getValue()`. Also within the class is a `float` representation of the value. In cases where it is not possible to represent the value as a `BigDecimal`, for example, if it is a `NaN` value, the value of the `BigDecimal` will be null, the `float` value will then be relevant and can be retrieved by `getFloatValue()`. There is also a `string` representation stored of the original value, which can be retrieved from the method `getStringValue()`. The `toString` of this class will stringify the `BigDecimal` value whenever it is not null, otherwise it will stringify the `float` representation. There are also helper methods available such as `isNaN()` and `isInfinity()`.

See the Apama Javadoc listing for `com.apama.event.parser` for complete information about these classes. The Apama Javadoc documentation is available at *<apama_install_dir>*`\doc\javadoc\index.html`.

Apama .NET Client SDK

A new class, `DecimalFieldValue` has been added to represent the Decimal Field Type. This will always represent the decimal value within a Dot Net Decimal where possible, and is retrievable with calls to the property Value. Also within the class is a `float` representation of the value. In cases where it is not possible to represent the value as a Dot Net Decimal, for example, if it is a `NaN` value, the value of the Dot Net Decimal will be null, the `float` value will then be relevant and can be retrieved by the property `FloatValue`. There is also a `string` representation stored of the original value, which can be retrieved from the property `StringValue`. The `ToString` of this class will stringify the Dot Net Decimal value whenever it is not null, otherwise it will stringify the `float` representation. There are also helper properties available such as `IsNaN` and `IsInfinity`.

See the Apama .NET documentation for `Apama.Event.Parser Namespace` for complete information about these classes. The Apama .NET documentation is available at *<apama_install_dir>*`\doc\dotNet\engine_client_dotnet5.0.chm`.

Apama C/C++ Client SDK

Because there is no built-in support for the decimal type in C and C++, decimal values passed back over the Apama C event parser API are encoded as 64-bit binary integer decimals, as described in IEEE 754-2008. For Apama applications that need precise decimal manipulation, Apama recommends you use a third-party library, such as the freely available Intel Decimal FP library.

New ADBC adapter features

In Apama 5.0, the ADBC adapter has been enhanced in a variety of areas.

Some of these improvements change the signatures and/or behaviors of several actions. Applications that are being migrated from Apama 4.3 will need to be modified to use the new signatures, which are fully described in "Using the Apama Database Connector" in *Connecting Apama Applications to External Components*. In addition, to migrate Apama 4.x projects that use the ADBC Common bundle, you need to manually update the Apama 4.x projects.

General API improvements

In earlier releases, the settings for `_ADBCTable` when storing events or data and `_ADBCTime` when storing events were specified in an action's `extraParams` parameter. With release 5.0 you now use the `tableName` parameter when storing events and data and the `timeColumn` parameter when storing events to specify this information. This change affects the following ADBC actions:

- `Connection.storeEvent`
- `Connection.storeEventWithAck`
- `Connection.storeData`
- `Connection.storeDataWithAck`
- `Connection.storeDataWithAckId`

In earlier releases, the settings for `_ADBCType` and `_ADBCTime` when running playback queries were specified in the `extraParams` parameter. With release 5.0, you use the following setter actions to specify this information:

- `Query.setEventType`
- `Query.setTimeColumn`

Note: The setter actions are not necessary when using the ADBC Sim adapter for playback.

The seldom used `uniqueConnection` parameter is now specified in the `extraParams` in the following action:

- `Connection.openDatabaseFull`

The following actions now return an `integer` id. This makes them consistent with the `commitRequest` actions.

- `Connection.rollbackRequest`
- `Connection.rollbackRequestFull`

Token parameter added

In release 5.0, a new string `token` parameter has been added to selected actions and callbacks to permit matching in the callback. This allows the callback to perform different operations depending on the token value. This eliminates the need to create different callbacks. The following actions use the `token` parameter.

- `Connection.storeEventWithAck`
- `Connection.storeDataWithAck`
- `Connection.storeDataWithAckId`
- `Connection.commitRequestFull`
- `Connection.runCommand`
- `Connection.runCommandFull`
- `Query.setBatchDoneCallback`
- `Query.setResultEventCallback`

A SQL statement can be specified for store actions

With ADBC store operations, you can now use either a provided SQL statement or the generated statement. The SQL statement provided can perform an `insert` or `update` and the use of a stored procedure is supported.

- `Connection.storeEvent`
- `Connection.storeEventWithAck`
- `Connection.storeData`
- `Connection.storeDataWithAck`
- `Connection.storeDataWithAckId`

The following actions have been added to create and delete `storeStatements` used with the above actions

- `Connection.createStoreStatement`
- `Connection.deleteStoreStatement`

New FixedSizeInsertBuffers property

The `FixedSizeInsertBuffers` is a new property that allows you to change the default buffer size used when the `StoreData` and `StoreEvent` actions perform batch inserts. The property is specific to ODBC. Apama uses the `FixedSizeInsertBuffers` property to specify the size of the buffers for the columns. The default "true" uses a fixed buffer size of 10K bytes for each column. If the value is changed to "false", the size of the column buffers is determined dynamically by examining the database table into which the data will be inserted. Allowing the buffer size to be set dynamically can significantly reduce memory usage when performing batch inserts to database tables that contain hundreds of columns or when using a very large `StoreBatchSize`.

Database connections can be opened in read-only mode

A parameter has been added to specify that a database connection should be opened in read-only mode. When a connection is opened in read-only mode an error will be reported for any API action that requires writes (`Store`, `Commit`, or `Rollback`). Most databases do not prevent writes from a connection in read-only mode so it is still possible to perform writes using the `Command` actions. The adapter will log a message to this effect for connections opened in read-only mode.

The open actions that use the `readOnly` parameter are:

- `com.apama.database.Connection.openFull`
- `com.apama.database.Connection.openShared`
- `com.apama.database.DBUtil.open`
- `com.apama.database.DBUtil.openShared`

New open "shared" actions

Both the ADBC Event API and the ADBCHelper API have new open "shared" actions that will use an already open connection if one is found; if the action does not find a matching open connection, it opens a new one. The actions are:

- `com.apama.database.DBUtil.openShared` is part of the ADBCHelper API
- `com.apama.database.Connection.openDatabaseShared` is part of the ADBC Event API

New reconnection capability

Apama applications can automatically reconnect if a disconnection error is encountered. The reconnection capability is optional and the default is to not reconnect when a disconnection error occurs. The following reconnection actions are available to control the reconnection behavior.

- `action setReconnectPolicy()`
- `action setReconnectTimeout()`

Need to manually add the ADBC Common bundle to a project

Existing Apama 4.x projects that use the ADBC adapter require manual updating when used with Apama 5.*x* . This is necessary in order to update bundles that contain static files. To perform this update, you need to perform the following:

- In the Project Explorer, right-click the project and select **Apama Build Path > Configure Build Path**.
- In the Properties window select the **Bundles** tab.
- Select the **Add** button, then the "ADBC Adapter Common" bundle listed under the Adapter bundles, then **OK**.
- Select the **OK** button on the Properties window.

The ADBC adapter static files will now be updated to the 5.*x* version.

Packaged Progress database drivers

The Apama 5.0 release includes Progress DataDirect ODBC and JDBC database drivers for the following Apama-certified databases (note, the ODBC drivers are available only for Windows platforms):

- DB2
- Microsoft SQL Server
- Oracle

These database drivers eliminate the need to install vendor-supplied drivers and they are pre-configured so adapter instances are automatically configured with appropriate settings when the drivers are added to an Apama project.

The Progress ODBC database drivers are licensed to be used only with the Apama ADBC adapter. The Progress JDBC drivers can be used with any Apama component.

For more information on the supplied Progress database drivers, see the Progress DataDirect Connect Series documentation available in the following locations:

```
<apama_install_dir>\doc\db_drivers\jdbc
<apama_install_dir>\doc\db_drivers\odbc.
```

See also: "Upgrading to Apama-supplied JDBC and ODBC database drivers in 5.2 " on page 65.

New Web Services client adapter features

In Apama 5.0, the Web Services client adapter has been enhanced as follows:

- You can add multiple Web Services Client adapter instances to an Apama project. Apama Studio generates service monitors and configuration files that are specific to each adapter instance.
- The Web Services client adapter supports using multiple threads for sending input messages. In some situations an application may experience throughput performance gains when using multiple threads.
- Many limitations on convention-based mapping have been removed. For example
  - Element names containing hyphens and dots can be represented using `$002d` and `$002e`, respectively.
  - Mapping both text and attribute/child node data from the same XML node is supported.
  - Mapping attribute values to EPL `integer`, `float`, and `boolean` fields is supported.

See also: "Migrating 4.3 projects that use Web Services Client adapters" on page 118.

For complete information, see "Using the Apama Web Services Client Adapter" in *Connecting Apama Applications to External Components*.

New correlator-integrated messaging for JMS features

In Apama 5.0, the correlator-integrated messaging for JMS feature has been enhanced as follows:

- In addition to specifying static senders and receivers in the adapter's configuration file, Apama has a new event API that allows senders and receivers to be added and removed dynamically from EPL code. This provides the ability to receive from a topic or queue, the name of which is known only at runtime. For advanced users, it provides a way to allow an application to implement dynamic scaling of the number of sender and receiver threads in order to improve performance, based on any policy specific to the application's use case.

- JMS durable topic subscriptions are supported for dynamic and static receivers. This enables an Apama application to persistently register interest in a topic's messages with the JMS broker such that even if a correlator is down, messages sent to the queue will be held and be ready for delivery when the correlator recovers.
- Status notification events (such as `OK`, `CONNECTING`, `ERROR`, `FATAL_ERROR`, and `REMOVED`) are now sent for each JMS connection, receiver, and sender.
- The `maxExtraMappingThreads` property has been added to `ReceiverSettings`. This supports scaling receive-side mapping work (especially expensive XML parsing) across multiple cores without needing to add extra JMS receivers, which often adds a considerable extra burden, and breaks message ordering). By default it is set to 0 (that is, do mapping on the same thread as receiving). While this property should be used with extreme care, it provides a very useful tool for cases where performance needs to be improved.
- Receive message mapping conditional expression improvements:
  - The `jms.body.type` conditional expression has been added. This provides for discriminating between TextMessage, MapMessage, etc.
  - Added support for `empty` JUEL keyword, and checking the values of properties and headers that might not have been set.
- Periodic "JMS Status:" INFO log messages, similar to the existing correlator status log messages, are now logged for correlators with correlator-integrated messaging for JMS enabled. The JMS Status messages include diagnostic info such as number of outstanding sent events, totals send/received, used JVM memory, etc.
- A new receiver/sender settings property called `logPerformanceBreakdown` can be enabled to provide detailed information highlighting where performance bottlenecks may be occurring.
- The `JMSSender.getOutstandingEvents` action has been added to the event API. This can be useful for throttling the rate at which messages are emitted from an EPL application in order to avoid the JMS sender getting too far behind (and using up excessive memory as a result).
- A new `receiverFlowControl` property has been added to provide EPL applications with control over the rate at which events are taken from the JMS queue or topic by each JMS receiver.
- Many limitations on convention-based mapping have been removed. For example
  - Element names containing hyphens and dots can be represented using `$002d` and `$002e`, respectively.
  - Mapping both text and attribute/child node data from the same XML node is supported.

  - Mapping attribute values to EPL `integer`, `float`, and `boolean` fields is supported.

- Apama provides new sample applications that illustrate the use of correlator-integrated messaging for JMS. The *`apama_dir`*`\samples\correlator_jms` directory contains the following sample applications:

  - `simple-send-receive` - This application demonstrates simple sending and receiving. It sends a sample event to a JMS queue or topic as a JMS TextMessage using the automatically configured default sender and receives the message using a statically-configured receiver.

  - `dynamic-event-api` - This application demonstrates how to use the event API to dynamically add and remove JMS senders and receivers. In addition, it shows how to monitor senders and receivers for errors and availability.

  - `flow-control` - This application demonstrates how to use the event API to avoid sending events faster than JMS can cope with and a separate demonstration of how to avoid receiving messages from JMS faster than the EPL application can cope with.

For information on these features, see "Using correlator-integrated messaging for JMS" in *Connecting Apama Applications to External Components* and the ApamaDoc, which is available at *`apama_install_doc`*`\doc\ApamaDoc\index.html`

MIGRATION NOTES

If you are migrating an application from Apama 4.3 then the following changes to correlator-integrated messaging for JMS may require changes to your application.

- It is possible in some cases that customers may have created a configuration in which senders or receivers share the same receiver or sender id. As of Apama 5.0, this is no longer allowed, so any such ids should be renamed.

- The recommended `jms-global-spring.xml` file now uses bean classes `com.apama.correlator.jms.config.JmsReceiverSettings` and `com.apama.correlator.jms.config.SenderSettings` instead of `com.progress.adapters.config.jms.JmsReceiverSettings` and `com.progress.adapters.config.jms.JmsSenderSettings`. Apama recommends that customers manually edit the `jms-global-spring.xml` file and change the bean classes in order to benefit from new features.

- Various JMS status events are now enqueued to the correlator by the JMS runtime, which may result in '`Failed to parse the event`' WARN messages. Applications and deployment scripts that use correlator-integrated messaging for JMS should be updated to inject `CorrelatorJMSEvents.mon`. Applications developed within Apama Studio will automatically have this file accessible to them.

- The format of the `uniqueMessageId` that is generated by correlator-integrated messaging has changed since 4.3.4.0 (if not overridden explicitly with a user-supplied identifier). This means there is a slight chance that messages sent by 4.3.4.0 and redelivered after recovery in a 5.0 correlator might not be duplicate detected correctly by the downstream system. If this rare condition is a concern, ensure your application pauses emitting messages to JMS before beginning the upgrade.

- Direct use of the JMSPlugin in EPL code is now deprecated. Use the `com.apama.correlator.jms.JMS` event type instead.

Event parser usability/type-safety improvements

Apama's event parser library for Java has been improved to leverage Java generics and varargs to make it easier to use, and to provide automatic compile-time type safety for client applications that use the newly introduced generic methods and type parameters. The changes are:

- `Field` and `FieldType` now have a generic type parameter. This means that the compiler can assist with checking field types and eliminates the need for lots of type casts. The type parameter indicates the Java type used to represent field values, for example, for a sequence of integers the type parameter would be `List<Long>`.

  Note, this change will not break existing code, but will cause warnings until the generic parameters are added to the caller's code. You can also add the Java annotation `@SuppressWarnings("unchecked")` to suppress specific compiler warnings.

- Added new `getField` and `setField` method overloadings. These methods accept either a `Field<T>`, or a field name and `FieldType<T>` parameter. With this change, the methods take advantage of the generic type information to enforce correct parameter and return types at compile-time.

  Note, you can still use the old `getField(fieldName)` and `setField(fieldName)` methods to maintain backwards compatibility or if you are writing totally generic code that does not know what event types it is dealing with.

- The event parser library now explicitly prevents setting or getting null field values when using the new `getField` and `setField` methods. This simplifies coding and ensures you get a helpful message instead of a `NullPointerException` when getting a field value that was not assigned.

- The event parser library supports method chaining in `setField` to allow "fluent" coding style, allows fully initializing an event instance without needing to assign it to a temporary variable

- Added `FieldType.newField()`, `DictionaryFieldType.type()`, and `SequenceFieldType.type()` helper methods. These methods avoid the need to duplicate the generic type parameters on the RHS (as well as the LHS) when constructing fields and types. Using `fieldType.newField(name)` saves time and effort as compared to using `new Field<type parameter>(name, fieldType)`.

- Use varargs for `EventType(fields)` to produce shorter code

- Introduce a new, optional varargs parameter to the `EventParser` constructor to simplify the typical need to initialize the event parser with a set of event types on startup
- The Javadoc documentation has been updated and now contains an example that illustrates the new behavior of the `EventParser` class

Correlator Deployment Packages

Apama 5.0 introduces Correlator Deployment Packages (CDP). A CDP contains application EPL code in a proprietary, non-plain-text format. This treats EPL files similarly to Java files in a JAR file. You can inject a CDP file to the correlator just as you inject an EPL file or a JAR file containing a JMon application. When you inject a CDP file into the correlator, the package ensures that all the EPL files in the CDP are injected and injected in the correct order.

To support Correlator Deployment Packages, the following features have been added to Apama:

- The `engine_package` utility is a new command line tool in Apama 5.0. The utility assembles EPL files into a Correlator Deployment Package (CDP).

  For more information on `engine_package`, see "Correlator Utilities Reference" in *Deploying and Managing Apama Applications*.
- The `engine_inject` utility has a new `-c` (`--cdp`) option that is used to inject CDP files.

  For more information on `engine_inject`, see "Correlator Utilities Reference" in *Deploying and Managing Apama Applications*.
- The Apama C/C++ API has the following new methods for injecting CDP files:
  - `virtual void injectCDP(const AP_uint8* cdpbytes, AP_uint32 size, const char *filename=NULL) = 0;`
  - `virtual const char* const* injectCDPWithWarnings(const AP_uint8* cdpbytes, AP_uint32 size) = 0;`
  - `virtual const char* const* injectCDPWithWarningsFilename(const AP_uint8* cdpbytes, AP_uint32 size, const char *filename) = 0;`

  For more information on the C++/C API, see "Developing Custom Clients" in *Developing Apama Applications*.
- The Apama client API for Java has the following new methods for injecting CDP files:
  - `void injectCDP(byte[] cdpBytes)`
  - `void injectCDP(byte[] cdpBytes, java.lang.String filename)`

    - `void injectCDPsFromFile(java.util.List<java.lang.String> filenames)`

  For more information on the C++/C API, see "Developing Custom Clients" in *Developing Apama Applications*.

- The Apama client API for .NET has methods equivalent to the Apama client API for Java methods for injecting CDP files. For more information on the .NET client API, see the Apama .NET documentation help file located at `*install_dir*\doc\dotNet\engine_client_dotnet5.0.chm`
- In Apama Studio, you can use CDP files by adding them to your projects in the following ways:
    - As external dependencies to the **MonitorScript Build Path**.
    - Directly including them in a project, for example, by dragging the CDP file from Windows Explorer and dropping it onto the project in Apama Studio's **Project Explorer** view.

  For more information on adding resources to Apama projects, see "Working with Projects" in *Using the Apama Studio Development Environment* (renamed to *Using Apama with Software AG Designer* in version 9.9).
- Apama Studio provides a new feature to export EPL files as a CDP file.

  See "Exporting correlator deployment packages" in *Using the Apama Studio Development Environment* (renamed to *Using Apama with Software AG Designer* in version 9.9).

New correlator startup option

When you start the correlator you can now specify the `-Pclear` option to indicate that you want to clear the contents of the recovery datastore. This option applies to the recovery datastore you specify for the`-PstoreName` option or to the default `persistence.db` file if you do not specify the `-PstoreName` option. When the correlator starts it does not recover from the specified recovery datastore.

Changes to engine_send

In the Apama 5.0 release, the performance of `engine_send` has been improved to make it more suitable for performance benchmarks. In order to facilitate this a couple of features have been removed.

- The following arguments have been removed from `engine_send`:

  ```
  --doNotBatch | -d
  ```

  Now all event files will be automatically batched by `engine_send`.
- BATCH tags

BATCH tags without timestamps are now ignored by `engine_send`. Batching will be performed by `engine_send` and the client library. BATCH tags with timestamps will behave as they have in earlier Apama versions. All events in the previous batch will be flushed followed by a sleep until the next batch time.

- Multi-line, ml-style comments removed

  To simplify parsing of the event file in `engine_send/* ... */` comments are no longer supported in event files. This change is in order to improve the performance of `engine_send`.

  Comments may still be added as single-line comments with `//` or `#` comments. When editing event files in Apama Studio, multiple lines can be commented out using the 'Toggle Comment' action, which will prepend every line with `//`.

When using `engine_send`, remember to use UTF-8 input files and to send them with no conversion (using the `-u` or `--utf8` flags).

Deprecated serialize and deserialize operations have been removed

In Apama 5.0, the deprecated serialize and deserialize operations have been removed. This includes the following:

- Serialize and deserialize operations in the Apama C/C++ client software development kit.
- Dump and load operations in the Apama client software development kit for Java.
- Dump and load operations in the Apama .NET Engine Client.
- The `--initFrom` (`-C`) and `--initFromFile` options for starting the correlator.
- The `engine_dump` and `engine_load` correlator utilities have been removed.
- The Checkpoint tab has been removed from the Enterprise Management and Monitoring console (EMM).

MIGRATION NOTES

Applications that need to write their state to persistent storage might be able to use the Apama correlator persistence features. Correlator persistence allows an application to automatically write the state of the correlator to disk. When the correlator is shut down and then restarted, the correlator restores the most recently saved state.

For a brief description of the steps required to use correlator persistence, see the migration notes in "Deprecated high availability components removed" on page 107.

For complete information about Apama's correlator persistence, including how to design applications to use persistence-enabled correlators, how to fine-tune snapshot and recovery behavior, and descriptions of how correlator persistence interacts with other Apama components such as the MemoryStore and correlator plug-ins, see "Using Correlator Persistence" in *Developing Apama Applications*.

Deprecated high availability components removed

In Apama 5.0, high availability components that were deprecated have been removed. This includes the following:

- High availability correlators.
- Sender, merger, and sender/merger components.

Because you can no longer use high availability correlators, or sender, merger, and sender/merger components, the commands and icons for adding and managing those components have been removed from the Enterprise Management and Monitoring console (EMM).

MIGRATION NOTES

For applications in which it is unacceptable to lose any information, it may be appropriate to use Apama's correlator persistence feature. Correlator persistence automatically stores the state of the correlator runtime on disk. This means that when you stop and restart a correlator, the correlator restores the most recent saved state.

For complete information about Apama's correlator persistence, including how to design applications to use persistence-enabled correlators, how to fine-tune snapshot and recovery behavior, and descriptions of how correlator persistence interacts with other Apama components such as the MemoryStore and correlator plug-ins, see "Using Correlator Persistence" in *Developing Apama Applications*.

Briefly, to use correlator persistence, the following steps are needed.

- Specify the keyword `persistent` before the monitor declaration for each monitor written in EPL that you want to be persistent. For example:

  ```
  persistent monitor Order {
     action onload() {
     ...
     }
  }
  ```

- Optionally, define `onBeginRecovery()` and `onConcludeRecovery()` actions in your persistent monitors. The correlator executes these actions during the recovery process.
- Specify one or more persistence options when you start the correlator. You must always specify the `-P` option to enable correlator persistence.

  In addition there are several correlator options such as `-PsnapshotInterval` and `-PstoreLocation` that can be used to change the correlator's default behavior.

Changes in Apama Studio

Apama Studio now supports integration with an external Eclipse installation.

Apama profiler

The profiler for Apama applications has been completely rewritten and usability is significantly improved.

- The profiler no longer uses the Eclipse Test and Performance Tools Platform (TPTP).
- Now you can launch and profile the entire launch configuration in single operation.
- A launch shortcut available for the Apama project.
- You can take snapshots of data and compare the information in the snapshots with incoming, current data.
- More than one correlator of the project can be profiled in one shot.
- The new profiler is more lightweight.
- Columns view can be filtered, ordered, and rearranged.
- View filters are more generic.
- Differences between snapshot and incoming data are more cleanly displayed.

Event definitions from XML and XSD files

Apama event definitions can now be generated automatically from the structure of an XML document and from a schema specified in an XSD file. When you select the **New > New Event Definition** menu item, the New Event Definition dialog presents two additional choices, **XML File** and **XSD File**. When you select a valid XML or XSD file, Apama Studio creates an event definition from the file's XML structure or from the XSD schema.

Bundles removed

The following bundles are no longer needed and have been removed from Apama Studio:

- DatabaseSupport
- FinanceSupport
- SimpleExchangeSimulator

Changes related to dashboards

Changes related to dashboards are described below. For information about migrating dashboards, see the migration notes at the end of these descriptions.

Tomcat no longer bundled with Apama

The Tomcat application server and Tomcat-specific features have been removed from Apama.

Named Data Server support

Advanced users can now associate non-default Data Servers with specific attachments and commands. This provides additional scalability by allowing loads to be distributed among multiple servers. It is particularly useful for Display Server deployments. By deploying one or more Data Servers behind a Display Server, the labor of display building can be separated from the labor of data handling. The Display Server can be dedicated to building displays, while the overhead of data-handling is offloaded to Data Servers. For more information, see *Working with multiple Data Servers* in *Building and Using Dashboards*, *Managing the Dashboard Data Server and Display Server*.

Dashboard deployment changes

The Deploy/Publish wizard has been streamlined and simplified. For more information, see "Using the Deployment Configuration editor" in "Preparing Dashboards for Deployment" in *Building and Using Dashboards* as well as "Deploying Dashboards" also in *Building and Using Dashboards*.

Dashboard message localization support

For thin-client (Display Server) deployments, you can now localize the text displayed in pop-up menus, login windows, status windows, and various error messages. For more information, see "Localizing dashboard messages" in *Building Dashboard Clients*.

Dashboard attachment and command changes

The command and attachment dialogs now display the scenario ID for selection, not the display name. This resolves ambiguity if two scenarios or DataViews have the same display name. This change is backwards compatible.

New dashboards panels options

The dashboards panels layout definition file, panels.ini, has been enhanced to provide more flexibility when creating multi-panel displays. This includes the ability to have resizeable panels.

Enhanced tree control

The tree control lets you create a rich and compact visual presentation of hierarchical data, often in a multi-panel application for display navigation. With Apama 5.0, the tree control also lets you use this type of display when the number of nodes in the tree is dynamically changing or when the state of a node changes. For more information, see *Building Dashboard Clients*.

Fx objects removed

Builder's **Fx** tab has been removed. You must eliminate Fx objects from any dashboards that use them. In most cases replacement visualizations are available for Fx objects.

Data Server status table change

The Data Server status table now contains an additional column, **Config**, that contains a string identifying the Data Server version. See "Dashboard data tables" in *Building Dashboard Clients*.

New and changed dashboard functions

The functions listed below have been added or changed. For more information, see the *Dashboard Function Reference*.

- Concatenate Columns
- Delta And Rate Rows
- Ensure Columns
- Ensure Timestamp Column
- Format Table Columns
- Group By Time
- Group By Time And Unique Values
- Group By Unique Values
- Join Outer
- Rename Columns
- Table Contains Values
- Validate Substitutions

New Custom command

The new set substitution command provides a way to set substitution values without going through a drilldown (via the original **Drill Down or Set Substitution** system command).

To use this command, right click the command property and select **System**. In the **Define System Command** dialog, in the **Command Type** combination box, select **Execute Custom Command**. Type `Apama_SetSub1.0` in the **Command Name:** text box.

In the **Command Value:** field, enter the string:

```
Sub=Value[;Sub=Value...]
```

For example, to set `$MySub1` to `value1` and `$MySub2` to `value2`, enter this command value:

```
MySub1=value1;MySub2=value2
```

Note that the `$` is removed from the substitution name.

New and changed visualization objects

The following visualization objects have been added or changed:

- Button control: The `borderWidth` property has been removed.
- Indicator objects: All of the indicators previously on the **Indicators** tab of the Object Palette have been replaced with the following four new indicators: `obj_ind_discrete` (supports three discrete comparisons), `obj_ind_limits` (supports two high and two low thresholds), `obj_ind_multi` (supports an unlimited number of comparison values; for each comparison, you can specify whether the value property must be equal to, not equal to, greater than, or less than the comparison value), `obj_ind_panel` (a panel of three indicator lights; each indicator light supports a discrete comparison). For all indicators, attach the value property to data, and set up your comparisons using the properties in the **Alert** category. The old indicators in existing displays will continue to work as they did before.
- Objects with `image` properties: Builder now includes a library of images that you can use on any object that supports images. You can also define a custom image library. To access the images, edit the `image` property on an object. Instead of entering an image name, click on the **...** button, which brings up a dialog with a tree on the left. A preview of the selected image appears in the pane to the right. Note that you can still simply type your image name into the `image` property field instead of using the dialog. You can access the same image editor dialog from the **Image Name** field in the **Background Properties** dialog, as well as when editing images in the `filterProperties` property on the table object.
- Text entry field, text area, and password field: These objects have been enhanced so that when their value is longer than the control can display, the beginning of the string is displayed instead of the end. (This was already true for Display Server (thin client) deployments; the enhancement applies to other Web-base deployments, as well as Viewer and Builder.)
- Numeric text entry controls: These objects have been enhanced to support a new validation option for blank entries. To enable this feature, select the `validateBlankValuesFlag`. If selected and the user enters a blank string, the `actionCommand` will not execute and the `invalidInputVarToSet` and `invalidInputMsgVarToSet` will be updated to indicate an invalid entry.
- Slider control: A new property, `axisDirection`, allows you to set the axis direction of the scale to the following: **Bottom to Top** (vertical orientation with the minimum on the bottom and the maximum on top), **Left to Right** (horizontal orientation with the minimum on the left and the maximum on the right), **Top to Bottom** (vertical orientation with the minimum on the top and the maximum on the bottom), and **Right to Left** (horizontal orientation with the minimum on the right and the maximum on the left). Another new property, `fgColor`, sets the tick mark color. You can either select a color or accept the default color. The `fgColor` is only applied when the control is enabled, so tick marks on sliders in the main Builder window will not use

it (preview your display to see the `fgColor` applied). This change does not apply to Display Server (thin client) deployments, whose sliders do not have tick marks.

- Date chooser control: This is enhanced to support three new properties: `fgColor` (the font color of the text entry area; default is black), `validColor` (the font color to use while editing in the text entry area if the entered value is using the correct format; default is green), and `invalidColor` (the font color used during and after editing in the text entry area if the entered value is not using the correct format; default is red). `validColor` and `invalidColor` are not supported in Display Server (thin client) deployments.
- Trend graph: A new property, `historyOnlyFlag`, controls whether only historical data is included. See "Dashboard Property Reference for Graphs, Tables and Trends" in *Building and Using Dashboards* for more information.

MIGRATION NOTES

The Dashboard Builder's **Fx** tab has been removed and Fx widgets are no longer available. You need to replace Fx objects in Apama project dashboards with equivalent visualization objects. Fx widgets in existing dashboard files will be ignored by dashboard processes.

Apama installations no longer include the Apache Tomcat application server. Apama dashboards can be deployed to these supported application server containers:

- Apache Tomcat
- WebSphere
- WebLogic

For an up-to-date listing of supported application servers, see the Software AG Knowledge Center in Empower at https://empower.softwareag.com/KnowledgeCenter/default.asp.

The Apama Web "Login" and "Menu" applications have been removed. For supported application servers, this logic needs to be implemented by the user.

Event Modeler standard block changes

In previous releases, Event Modeler included a number of standard blocks related to capital market applications. These blocks are no longer provided with Event Modeler. The blocks that have been moved are listed below:

- Basket Calculator
- EWMA Calculator
- MACD Calculator
- Market Depth
- Multi-Leg Order Manager

- OBV Calculator
- Order Flow
- Order Manager
- P&L Calculator
- Position Calculator
- RSI Calculator
- Volume Distributor
- VWAP Calculator

Also, the Database Storage and Database Retrieval blocks have been removed.

MIGRATION NOTES

Apama Capital Markets Foundation includes the removed blocks as well as these supporting EPL files:

- `TickManagerSupport.mon`
- `OrderManagerSupport.mon`
- `MultiLegOrderManagerSupport.mon`

In place of the Database Storage and Database Retrieval blocks, use the ADBC Storage and ADBC Retrieval blocks, which were added in a previous release.

Migrating from Apama 4.3

If you are upgrading from Apama 4.3 the following topics provide migration notes, information about backwards incompatible changes, and additional helpful information for migrating applications from Apama 4.3.

- "Changes to EPL features in Apama 5.0" on page 88
- "Deprecated serialize and deserialize operations have been removed" on page 106
- "Deprecated high availability components removed" on page 107
- "Event Modeler standard block changes" on page 112
- "Changes related to dashboards" on page 108
- "New correlator-integrated messaging for JMS features" on page 100
- "Changes to engine_send" on page 105
- "New ADBC adapter features" on page 96

Before migrating your application, read the information in the above topics as well as the information provided in this topic and in the following topics:

In Apama 5.0, the OEM installer has been removed. The silent installation mode has been retained. Information for creating a properties file and for running an installation in silent mode can be found in the Software AG Knowledge Center in Empower: http://empower.softwareag.com.

The Apama correlator needs a license file in order to run for more than 30 minutes. Previously the Apama product used a British spelling ("licence" - with a 'c'). For consistency with other products, this has been changed to the American spelling "license" (with an 's'). This change is visible in a number of places such as the correlator command line, sentinel agent command line, EMM. If your scripts use the old spelling, you should modify them to use the new spelling. Note that the this affects the `--license` option for starting the correlator and the sentinel agent. (Scripts that use the short-hand command-line parameter to the correlator "`-l`" do not need to change.) This also affects the Ant properties and task attributes in `apama-macros.xml`, so Ant scripts and properties files may no longer run until the spelling is updated.

The environment variables `APAMA_AGENT_CWD` and `APAMA_AGENT_ARGS` no longer have any relevance on UNIX. Users should either use the default settings, run `sentinel_agent` by hand, or edit the `init` script.

Apama no longer supports Progress Control Tower.

For further migration assistance, contact Apama technical support.

Platform changes between 4.3 and 5.0

In Apama 5.0, there are significant changes to various platforms. Details about supported operating systems, Eclipse versions, Java releases, application servers, JMS providers, databases, and web browsers are available in Empower. The following platforms have been removed:

Operating systems

- Windows Server 2003 R2 32-bit, Windows Server 2003 64-bit, Vista, and XP are no longer supported.
- Linux 32-bit platforms are no longer supported.

Eclipse versions

Eclipse 3.5 and Eclipse 3.6 support has been removed.

Java support

JVM 1.5 is no longer supported.

JEE Application Servers

- WebSphere Application Server 6.x is no longer supported.
- JBoss Application Server is no longer supported.
- Apama 5.0 does not ship any JEE Web Container.

JMS buses and providers

- WebSphere Application Server 6.x is no longer supported.
- JBoss Enterprise Application Platform (JBoss messaging) is no longer supported.
- Sonic 7.x and 8.0.x are no longer supported.

Databases

- Microsoft SQL Server 2008 is no longer supported.
- OpenEdge is no longer supported.
- Oracle RAC 11g R2, RAC 10g R2 and 10g R2 releases are no longer supported.

Web browsers

Internet Explorer 7 is no longer supported.

Changes to Apama client APIs between 4.3 and 5.0

This topic describes various changes to the Apama client APIs.

Event Parser API for Java generics warnings

In the Apama `EventParser` API for Java, the `Field` and `FieldType` classes now have a generic type parameter. This means that the compiler can assist with checking field types and eliminates the need for lots of type casts. The type parameter indicates the Java type used to represent field values, for example, for a sequence of integers the type parameter would be `List<Long>`.

Note, this change will not break existing code, but will cause warnings until the code is updated to specify the generic type parameters. You can also add the `@SuppressWarnings("unchecked")` annotation.

Event Parser removed methods

In the Apama API for Java, the following methods have been removed:

- `SequenceFieldType.setElementType`
- `DictionaryFieldType.setKeyType`
- `DictionaryFieldType.setValueType`

This means the types must be specified in the constructors. When creating events with sequences or dictionaries of that type, create the event type first, then the sequence or dictionary type, and then add that field to the EventType.

Similarly, in the .NET API, the `SequenceFieldType.elementType`, `DictionaryFieldType.keyType`, and `DictionaryFieldType.valueType` elements are now read-only.

Change in EventService and ScenarioService Destroy() and Dispose() semantics

The semantics of `IEventService.destroy()` and `IScenarioService.destroy()` have been changed and simplified. The caller is now responsible for destroying an event service or client bean passed into `EventServiceFactory` or `ScenarioServiceFactory`, respectively. The caller is never responsible for destroying any event service or client object implicitly created by the constructor as this will happen automatically. Apama recommends that you review existing code to check for compliance with these semantics, ensuring that the objects are correctly shut down. The `Dispose()` and `Destroy()` methods are now idempotent, so it is safe to call them repeatedly.

The `IEventServiceChannel.destroy()` method has been removed; the correct way to remove an event service channel is still to use the `IEventService.removeChannel()` method.

Change in engine client semantics

The following changes have been made to the semantics of the client API stacks for Java and .NET.

- The `ALL_CONSUMERS_CONNECTED` engine client property semantics have changed such that it will now always have the value 'false' if the bean itself is disconnected, even if no consumers are registered.
- The engine client (`EngineClientBean`) layer no longer guarantees order of event delivery for synchronous consumers across all engine client objects in the JVM. However, event delivery order is still guaranteed for each engine client object (bean) and this change is unlikely to have an impact on existing applications.
- The EventService's reconnection interval is now identical to the engine client (that is, the beans layer) connection polling interval, instead of being hardcoded to 500ms. However, this interval respects any reconnection period limit specified by the engine client's `setReconnectPeriod()` method.

Removal of deprecated items from APIs for Java and .NET

Removed several methods and classes from the APIs for .NET and Java that were deprecated or extremely unlikely to be used in customer code. The removed .NET methods are:

- `Apama.EngineException.getWarnings`
- `Apama.Event.Parser.FieldTypeFactory`
- `Apama.Event.Parser.ParserRuntimeException.#ctor`

- `Apama.Net.Client.IBaseClient.EngineManagement`
- `Apama.Net.Client.IBaseClient.ReleaseEngineManagement(Apama.Engine.EngineManagement)`
- `Apama.Util.Logger.WriteStackTrace(System.Exception,System.IO.TextWriter)`
- `Apama.Util.Logger.BaseSuppressConfigName` and `LogSuppressPropertyName` constants.

The following Java methods and classes were removed:

- `com.apama.event.parser.FieldTypeFactory`
- `com.apama.event.parser.ParserRuntimeException`
- `com.apama.engine.beans.AbstractEngineClientBean`
- `com.apama.event.EventReader.CommentStripper`

Default batching/non batching semantic of the sendEvents was changed

The default semantic of the `sendEvents` calls has been changed from non-batching to batching in all remote client APIs. A non batching version called `sendEventsNoBatching` has been added to the `EngineManagement` interface together with a `flushEvents` method. The `sendEventsBatching` method has been removed.

C client API changes

The signatures of `attachAsEventConsumerTo` and `detachAsEventConsumerFrom` were changed. The `attachAsEventConsumerTo` has two new arguments: `persistent` and `disconnectSlow`. The `detachAsEventConsumerFrom` has one new argument: `persistent`.

The new signatures are:

```
AP_bool (AP_ENGINE_CLIENT_CALL* attachAsEventConsumerTo)(
   AP_EngineManagement* engine,
   AP_EngineManagement* target,
   const AP_char8* const* channels,
   AP_bool persistent,
   AP_bool disconnectSlow);
void (AP_ENGINE_CLIENT_CALL* detachAsEventConsumerFrom)(
  AP_EngineManagement* engine,
  AP_EngineManagement* target,
  const AP_char8* const* channels,
  AP_bool persistent);
```

Java classes built with Java 1.6

It is no longer possible to use a 1.5 JDK with Apama.

Remove Applet factory methods, constructors and getters and setters in API for Java

For historical reasons (CORBA required it) the constructors and factory methods for creating connections to other Apama systems have included an Applet parameter. This is no longer necessary and these APIs have been deprecated since 4.0. In Apama 5.0, these methods have been removed. This should not affect customer code.

Migrating 4.3 projects that use Web Services Client adapters

In Apama 5.0, to use Apama projects that were created using the Web Services Client adapter in Apama 4.3.4, you need to perform the following migration steps.

See also: "New Web Services client adapter features" on page 100.

1. Import the release 4.3.4 project into a new Apama 5.0 workspace or open an existing 4.3.4 workspace in Apama Studio. When first accessed Apama Studio will report some errors in the projects.
2. To eliminate these errors, do the following:
   a. Right-click the project name and select **Properties** from the pop-up menu. The Properties dialog appears.

   b. In the Properties dialog, select **MonitorScript Build Path** and select the **Bundles** tab.

   c. Select the `WebServices Client Adapter` entry and click the **Update** button.

      This updates the WebServices adapter bundle.

   d. When you click the **Update** button, Studio asks if you want to “Backup the existing files”. If you select **Yes**, Studio will create backup property and configuration files under the *`<apama_project_name>`*`/bundle_instance_files` project node.
3. If Apama Studio still reports errors in the project, refer to “Updating EPL to interact with WebServices”.

Updating EPL to interact with WebServices

In some situations, you may need to modify your existing code to be compatible with changes introduced in the Web Services Client adapter in release 5.0.

- Apama 5.0 only support requests and responses from public contexts. If your application needs to make requests from a private context, you need to write appropriate EPL code in order to make requests and listen for responses.
- Service monitors no longer support WebService timeouts; these need to be handled by EPL code in your Apama application.

After the modifications listed above have been made to an existing 4.x Apama project, the project can be run successfully in Apama 5.0.

Note: In this release, `WSResponse`, `SetupContextListeners` and `TerminateContextListeners` events are deprecated.

Change to XML parser

Apama 5.0 uses Saxon to parse XPath expressions unlike earlier releases of Apama that used Xerces. If you use XPath queries to fetch the text content of an XML element, you need to explicitly use the `text()` function at the end of your XPath query. If you do not use the `text()` function, the entire element will be returned.