Release Notes

Innovation Release

Version 10.0

April 2017

APAMA

This document applies to Apama Version 10.0 and to all subsequent releases.

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

Document ID: PAM-RN-100-20170419

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 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/apama-onlinehelp` 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 Software AG Designer 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, and Java. You can use one or several of these

Title	Description
	technologies to implement a single Apama application. In addition, there are 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.
Building and Using Dashboards	Describes how to build and use an Apama dashboard, which provides the ability to view and interact with data views. 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 or Web Start 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 Software AG Command Central. 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). ■ 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).
- Reliable messaging with connectivity plug-ins is not permitted.
- 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 10.0

Apama 10.0 is the successor of Apama 9.12.

Apama 10.0 runs on the platforms listed in the *Supported Platforms* document for version 10.0. 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 10.0 includes new features, enhancements, and changes as described in the following topics.

New HTTP client transport connectivity plug-in

A new connectivity plug-in, the HTTP client transport, is now available. It can be used to connect to external services over HTTP/REST, perform requests on them and return the response as an event. For detailed information, see "The HTTP Client Transport Connectivity Plug-in" in *Connecting Apama Applications to External Components*.

Using Software AG Designer, you can add the new HTTP client connectivity bundle to your projects. For further information, see "Adding the HTTP client connectivity plug-in to a project" in *Using Apama with Software AG Designer*.

This HTTP client transport connectivity plug-in is also delivered with version 9.12.0.5. In addition to the functionality provided with that fix, version 10.0 provides the following enhancements:

- The HTTP client can now encode query parameters from a map provided in the metadata of requests.
- The HTTP client now uses nested maps in metadata for metadata keys containing a period (.). For example, `metadata.http` is now a map containing `statusCode`, `statusReason`, `path`, and so on. Applications using the Mapper codec to set or read these will continue to work without change. Custom codecs may need changes to support this.

Apama integration with MQTT in 10.0

A new standard connectivity plug-in, the MQTT transport, is now available. It can be used to communicate between the correlator and an MQTT broker. This implementation of the MQTT transport is based on the Eclipse Paho C library which supports the MQTT v3 protocol. For detailed information, see "The MQTT Transport Connectivity Plug-in" in *Connecting Apama Applications to External Components*.

Using Software AG Designer, you can add the new MQTT connectivity bundle to your projects. For further information, see "Adding the MQTT connectivity plug-in to a project" in *Using Apama with Software AG Designer*.

Apama integration with Kafka in 10.0

A new standard connectivity plug-in, the Kafka transport, is now available. It can be used to communicate with a Kafka distributed streaming platform. In this release, it is provided as a sample in the samples/connectivity_plugin/java/KafkaTransport directory of your Apama installation. See also "The Kafka Transport Connectivity Plug-in" in *Connecting Apama Applications to External Components*.

Connectivity plug-ins enhancements in 10.0

Apama 10.0 includes the following connectivity plug-ins enhancements.

Reliable messaging

- Connectivity plug-ins can now perform reliable messaging. If you have done `APP_CONTROLLED` reliable messaging with the Java Message Service (JMS) before, then doing the same with connectivity plug-ins will look very similar. Your old way of doing acknowledgments and flushes in EPL will be quite easy to translate across to the new way of doing this with connectivity plug-ins. For more information, see "Using reliable transports" in *Connecting Apama Applications to External Components*.

  If you want to write your own transport that supports reliable messaging, see "Developing reliable transports" in *Connecting Apama Applications to External Components*.

- The Digital Event Services transport connectivity plug-in now supports reliable sending and receiving of events. For more information, see "Reliable messaging with Digital Event Services" in *Connecting Apama Applications to External Components*.

- A new metadata value, `sag.messageId`, is now available. This is used for reliable receiving. For more information, see "Metadata values" in *Connecting Apama Applications to External Components*.

User-defined status reporting

Connectivity plug-ins can now add any number of user-defined status values which are reported as part of the correlator's status information from the REST API, the `engine_watch` tool, the EngineClient API, and from the EPL Management interface. If the status keys follow the conventions listed in "Monitoring the KPIs for EPL applications and connectivity plug-ins" in *Deploying and Managing Apama Applications*, the status and KPIs of your application's connectivity plug-ins can be displayed by Command Central. For more information about how to report status, see "User-defined status reporting from connectivity plug-ins" in *Connecting Apama Applications to External Components*. See also the samples\connectivity_plugin\cpp\httpclient and samples\connectivity_plugin\java\HTTPServer directories of your Apama installation for examples of how to report status information from a connectivity plug-in.

Message metadata support for non-string values

The message metadata can now also store non-string values. With previous versions, it was allowed to only store string keys and values.

In both C++ and Java `Message` implementations, the `getMetadataMap()` function has been added which returns a view on the metadata which can contain non-string values (`map_t` for C++ and `Map<String,Object>` for Java). The `getMetadata()` function still exists, but is deprecated.

While the changes for Java maintain backwards compatibility for users of the `getMetadata()` function, the changes for C++ do not. For C++, the `metadata_t` iterator's `key()` and `value()` functions have been changed to return `std::string` instead of `const char*`. Only stringifiable values (bool, integer, double, decimal) can be obtained by calling `value()`, otherwise an exception is thrown. Moreover, for C++, the metadata `operator[]` has been changed such that it returns `std::string` instead of `const char*`. To access the full set of types in C++ metadata, use the new `getMetadataMap()` function instead.

For the existing connectivity plug-ins (such as the Mapper codec and Classifier codec), the assumption has been removed that only string keys and values can be stored into the metadata.

For more information, see "Metadata values" in *Connecting Apama Applications to External Components*.

New dynamic chain managers API

Connectivity chains can be created in three different ways:

- Statically using the `startChains` section of a YAML file.
- Dynamically by an explicit EPL call to `ConnectivityPlugins.createChain`.
- Dynamically by a connectivity plug-in. For example, a plug-in could create a chain in response to an EPL application subscribing or sending to a channel with a specific name or prefix.

In the previous release, it was not possible for user-defined connectivity plug-ins to dynamically manage chain creation, although some standard plug-ins shipped with the product (such as Universal Messaging and Digital Event Services) made use of this capability. In 10.0, there is a new "chain manager" API providing the ability for user-defined plug-ins written in Java to manage chains. In this release, it is not yet possible to do this from C++ plug-ins (it's only Java right now).

A transport plug-in that manages its own chains must provide a subclass of `com.softwareag.connectivity.chainmanagers.AbstractChainManager` which can create and destroy chains whenever it wishes, by instantiating any dynamic chain definition in the configuration files that contain the transport plug-in managed. A chain manager can register a listener that is notified whenever a new channel is used for the first time by the Apama EPL application, that is, when the application subscribes or

sends to a channel, which it can use to create a new chain to send/receive using that channel as needed.

For more information about configuring transport connectivity plug-ins that have a dynamic chain manager for use in your project, see "Configuration file for connectivity plug-ins" in *Connecting Apama Applications to External Components*.

For more information about how to create a dynamic chain manager transport plug-in, see "Static and dynamic connectivity chains" in *Connecting Apama Applications to External Components*.

Miscellaneous connectivity framework changes

- For Java, it is now also possible to set the classpath by defining a semicolon-delimited list of strings in the configuration file. See also "Configuration file for connectivity plug-ins" in *Connecting Apama Applications to External Components*.
- A new helper class called `com.softwareag.connectivity.util.MapExtractor` is available in the Java API. This class provides an easy and type-safe way to extract values from maps such as those used for plug-in configuration. It has automatic support for generating user-friendly error messages if configuration items are missing or of the wrong type. See the *API Reference for Java (Javadoc)* for further information.

Enhancements and additions to standard connectivity plug-ins

- A new standard connectivity plug-in, the String codec, is now available. It can be used to translate string payloads to binary payloads, and vice versa. For detailed information, see "The String codec connectivity plug-in" in *Connecting Apama Applications to External Components*.
- A C++ implementation of the JSON codec is now available. It behaves in the same way as the already existing Java implementation of that codec. For more information, see "The JSON codec connectivity plug-in" in *Connecting Apama Applications to External Components*.
- The JSON codec now has an additional configuration parameter `filterOnContentType` which allows it to ignore messages which are not encoded with the "application/json" content type. For more information, see "The JSON codec connectivity plug-in" in *Connecting Apama Applications to External Components*.
- The Mapper codec has been enhanced as follows:
  - With the `defaultValue` action, an empty or `null` value is now treated in the same way as an unset value.
  - It is now possible to set metadata or payload fields to lists or maps and top-level payloads to maps, in addition to setting them to strings.
  - A new `copyFrom` action is available. This new action is exactly the same as the `mapFrom` action except that the source field is not removed after the copy.

  For more information, see "The Mapper codec connectivity plug-in" in *Connecting Apama Applications to External Components*.

- The Mapper and Classifier codecs can now contain a period (.) for both payload and metadata rules. These correspond to nested maps. Mapper rules for built-in codecs and transports will continue to work without changes, but a mapper which is setting a metadata field containing a period will now produce a nested map structure. Custom codecs and transports may need changes to support this. See also "The Mapper codec connectivity plug-in" and "The Classifier codec connectivity plug-in" in *Connecting Apama Applications to External Components*.
- When mapping `nEventProperties` to metadata, the Universal Messaging transport connectivity plug-in now uses the prefix `um.properties`. You can use the Mapper codec to enable the original behavior. In addition, the ability to set `nEventProperties` while also sending a binary payload has been added. To do this, set `um.properties.`*`Key=Value`* in the metadata. See also "Supported payloads" in *Connecting Apama Applications to External Components*.
- With previous versions, only the host that created a Universal Messaging channel was able to publish to the channel created by Apama. As of this version, it is possible to specify ACL permissions for channels that are automatically created by Apama. This can be done by specifying the new `createChannelPermissions` option in the YAML configuration file for the Universal Messaging connectivity plug-in. See "Configuring the dynamicChainManagers" in *Connecting Apama Applications to External Components*. This behavior has also been changed in the deprecated native Universal Messaging integration (see also "Enabling automatic creation of Universal Messaging channels" in *Connecting Apama Applications to External Components*).

New API for writing EPL plug-ins in C++ in 10.0

In 10.0, the API for writing EPL plug-ins in C++ has been replaced. Support for the old C and C++ APIs defined in correlator_plugin.hpp and correlator_plugin.h is still retained in 10.0 for backwards compatibility but are deprecated and will be removed in a future release. Use of them will produce warnings both at compile time and when injecting monitors into the correlator which use a deprecated plug-in.

EPL plug-ins should now be written using the API in epl_plugin.hpp instead. See "Writing EPL Plug-ins in C++" in *Developing Apama Applications* for information on the new API.

For information on the differences between the new and old APIs, see "Comparison between the new and old APIs" in *Developing Apama Applications*.

For information on migrating plug-ins from the old APIs to the new one, see "Migrating from 9.12 or older plug-ins" in *Developing Apama Applications*.

The new EPL plug-in API does not include a pure-C API. Instead, the C++ API has been written with a compiler-agnostic ABI. This removes the requirement to use the platform compiler version when creating your plug-ins and therefore the need for a separate C API. All non-Java plug-ins should now be written against the new C++ API.

Apama enhancements in Software AG Designer 10.0

Apama 10.0 includes the following enhancements in Software AG Designer:

- The color in which annotations are shown in the EPL editor can now be configured in the Apama preferences. See also "Editor Colors" in *Using Apama with Software AG Designer*.
- The new field **Flush between events** has been added to the Data Player query page. It controls how many times the correlator's queues are flushed between every event from the database. This gives correct behavior in multi-context applications at the cost of speed. The default value for new queries is 2, which flushes the correlator's queues twice. Existing queries are not affected by this enhancement. A blank value is used in this case, which means that the correlator queues are not flushed. If you want to change this, you have to set the required value on the Data Player query page. See also "Specifying Data Player playback queries" in *Using Apama with Software AG Designer*.
- If a query pattern uses the `or` operator (see also "Query enhancements in 10.0" on page 19), then the New Query Send Event Action dialog includes the new **Required events** section. It sets which events must be matched by the pattern in order for the send event action to send the event. See also "Adding query send event actions" in *Using Apama with Software AG Designer*.
- The Open Type dialog functionality has been added for all Apama perspectives. See "Opening EPL elements and Java classes using Open Type dialog" in *Using Apama with Software AG Designer*.
- The Correlator Deployment Package wizard now accepts a launch configuration instead of a project. The ability to select the files to be included has been removed from the wizard. Instead, if you want to create a correlator deployment package (CDP) containing a subset of project's files, you must now create a dedicated launch configuration to represent the required files before opening the Correlator Deployment Package wizard. See "Exporting correlator deployment packages" in *Using Apama with Software AG Designer*.
- When you add a connectivity bundle, the standard-codecs.yaml file is now created in the **Linked Resources** node. This file contains standard codecs defined by Apama. See also "Connectivity bundles" in *Using Apama with Software AG Designer*.

Query enhancements in 10.0

Apama 10.0 includes the following query enhancements:

- A query may now use a decimal type for a parameter. See also "Format of query definitions" in *Developing Apama Applications*.
- In a `find` statement, you can now specify the `or` operator in the event pattern. The events on one side or the other of the `or` operator must be matched for the

query to fire. For detailed information, see "Query or operator" in *Developing Apama Applications*.

- Queries can now use the output of another query as input, thus allowing chaining of queries. For detailed information, see "Using the output of another query as query input" in *Developing Apama Applications*.
- To delete a query, you need to run `engine_delete -F` followed by a fully qualified query name. In previous releases, it was possible to delete a query by just using `engine_delete` if nothing else depended on it. Note that `engine_delete -F` deletes *everything* that depends on the query or the query output event. Therefore, caution is advised when deleting a query. See also "Deleting code from a correlator" in *Deploying and Managing Apama Applications*.

EPL enhancements in 10.0

Apama 10.0 includes the following EPL enhancements:

- The `throw` statement is now available. It causes an exception to be thrown. See also "The throw statement" in *Developing Apama Applications in EPL*.
- The behavior for disconnected out of band events has changed. In the `com.apama.oob` package, `componentName` and `address` have been added to `ReceiverDisconnected` and `SenderDisconnected` so that the behavior is now in line with `ReceiverConnected` and `SenderConnected`. A new `dictionary<string, string>` field called `extraParams` has been added to all the out of band events. Additionally in the logging, `componentName` has been changed to be "*`name`*", unless no name is provided in which case the old format "*`name (on port port_number)`*" is used. See also "Out of band connection notifications" in *Developing Apama Applications in EPL*.
- It is now possible to invoke EPL plug-ins written in Java from persistent monitors provided that they do not use chunks, which means that the behavior is now the same as for EPL plug-ins written in C++.

Correlator utility enhancements in 10.0

Apama 10.0 includes the following correlator utility enhancements:

- Instead of specifying a list of configuration files on the command line with the `--config` option, it is now also possible to specify these files in the `includes` section of another configuration file. For detailed information, see "Including YAML configuration files inside another YAML configuration file" in *Deploying and Managing Apama Applications*.
- It is now possible to specify the correlator port number in a YAML configuration file. For detailed information, see "Specifying the correlator port number" in *Deploying and Managing Apama Applications*.

- Using the new `engine_deploy` tool, it is now possible to generate an initialization file list, a correlator deployment package (CDP), or a correlator deployment directory from an Apama project that has been created with Software AG Designer. You can also use this tool to perform the initialization in a running correlator. If you are not using Software AG Designer, you can also use this tool with a directory of Apama files. For detailed information, see "Deploying a correlator" in *Deploying and Managing Apama Applications*.

  In addition, you can now also use Ant to generate a correlator deployment directory. The apama-macros.xml file includes the new `generate-correlator-deploy-dir` macro for this purpose. See also "About deploying Apama applications with an Ant script" in *Deploying and Managing Apama Applications*.

- A new management request has been added: `startInternalClock`. This is available on correlators via the `--doRequest` option of the `engine_management` tool (see "Shutting down and managing components" in *Deploying and Managing Apama Applications*) and via the apama-macros.xml Ant script (see also "About deploying Apama applications with an Ant script" in *Deploying and Managing Apama Applications*). This request takes a previously externally clocked correlator and starts to clock it internally.

Dashboard enhancements in 10.0

Apama 10.0 includes the following dashboard enhancements:

- The REST API now also supports the `PUT /logLevel` request for Dashboard Data Servers and Display Servers. See also "Managing and Monitoring over REST" in *Deploying and Managing Apama Applications*.

Miscellaneous enhancements and changes in 10.0

Apama 10.0 includes the following miscellaneous enhancements and changes:

- When using the IAF-based CSV codec plug-in (`JCSVCodec`), you can now enable Excel compatibility mode by setting the new `excelCompatible` property to `true` in the IAF configuration file. Double quotes are then used to match the behavior of Excel. By default, this new property is set to `false`. See also "The CSV codec IAF plug-in" in *Connecting Apama Applications to External Components*.

  In addition, you can also enable Excel compatibility mode when sending a configuration event from the transport that is communicating with the CSV codec using the method described in "Multiple configurations and the CSV codec" in *Connecting Apama Applications to External Components*.

- The REST API now also supports the `/info/stats` URI for the correlator and the IAF. It provides access to the same information as the `/correlator/status` and `/iaf/status` URIs. Previously, the `/info/stats` URI was only available for

Dashboard Data Servers and Display Servers. See also "Managing and Monitoring over REST" in *Deploying and Managing Apama Applications*.

- The new method `GenericComponentInfo.getDataValue()` has been added to the Apama API for Java. Using this method, you can easily look up a value from a `GenericComponentManagement` information category that has unique keys (that is, dictionary content) without having to iterate through every item. See the *API Reference for Java (Javadoc)* for further information.
- For applications that report status to Command Central, the recommended values for the `".status"` key are now `STARTING`, `ONLINE` or `FAILED` (instead of `ONLINE` and `OFFLINE`, although `OFFLINE` is still permitted for legacy applications). See also "Monitoring KPIs for EPL applications and connectivity plug-ins" in *Deploying and Managing Apama Applications*.
- A title can now be provided when generating ApamaDoc from an Ant script or from Software AG Designer. See also "Generating ApamaDoc from an Ant script" in *Developing Apama Applications* and "Exporting ApamaDoc" in *Using Apama with Software AG Designer*.
- The ATM Fraud queries sample has been moved to the demos folder of the Apama installation, and it is now available as a demo from the Welcome page. With previous versions, it was located in the samples\queries\ATM_Fraud_Sample folder.
- The demos and samples have been updated to use EPL instead of scenarios. Keep in mind that the Event Modeler is deprecated and that support for creating and deploying scenarios will be removed in a future release (see also "Removed and deprecated features in 9.12" on page 41).

Platform changes in 10.0

Apama 10.0 runs on the platforms listed in the *Supported Platforms* document for version 10.0. This is available from the following web page: "http://documentation.softwareag.com/apama/index.htm".

Check the above mentioned *Supported Platforms* document for details about the newer platform versions supported by Apama 10.0. Due to these upgrades, the following versions are no longer supported:

- Red Hat Enterprise Linux 6 update 1 (x86) 64-bit (replaced by support for a more recent version).
- CentOS Linux 6 update 1 (x86) 64-bit (replaced by support for a more recent version).
- Oracle Linux 6.5 (x86) 64-bit (replaced by support for a more recent version).
- Software AG Common Runtime 9.12 (replaced by support for a more recent version).
- Apache Tomcat 7.0.62 (replaced by support for a more recent version).

- Software AG Universal Messaging 9.12 (replaced by support for a more recent version).
- Software AG Universal Messaging Client 9.12 (replaced by support for a more recent version).
- Oracle Java 8 update 101 (replaced by support for a more recent version).
- Software AG Designer 9.12 (replaced by support for a more recent version).
- Google Chrome 44 (replaced by support for a more recent version).
- Firefox Mozilla 45 (replaced by support for a more recent version).
- Apple Safari (desktop) 9 (replaced by support for a more recent version).
- Apple Safari (mobile) iPad iOS 4 (replaced by support for a more recent version).
- VMWare ESXi 5.0 (replaced by support for a more recent version).
- Software AG Terracotta BigMemory Max 4.3.3 (replaced by support for a more recent version).
- MathWorks MATLAB 2014b (replaced by support for a more recent version).

Windows compiler update in 10.0

Apama has upgraded from Microsoft Visual Studio 2013 (Visual C++ v12) to Visual Studio 2015 Update 3 (Visual C++ v14). Consequently, you must rebuild the following components with the new compiler:

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

In addition, when planning migration, consider that the C++ compiler in Visual Studio 2015 is more modern than the previous version and 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.

For example, one of our samples previously included this macro definition:

```
#define snprintf _snprintf
```

This produces a compilation error with the new compiler. So if your code includes this line, you should simply remove it.

Note: Our 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.

Miscellaneous changes in 10.0 affecting backwards compatibility

The following changes in Apama 10.0 affect backwards compatibility to previous Apama versions:

- Apama 10.0 incorporates the ICU (International Components for Unicode) Timezone Data update 2016j, which is the most recent update at the time of release. This will update timezone data used by the correlator and Time Format EPL plug-in.

  In addition, Apama 10.0 contains an upgrade from ICU 3.8.1 to ICU 58.1 which contains some small behavioral differences which may affect your applications:

  - When formatting time in UTC, the output for a "GMT+offset" timezone is now "GMT" rather than "GMT+00:00" (non-UTC timezones are unaffected).
  - The format/parse symbol `z` no longer uses timezone short codes such as "EST" due to ambiguity. It now uses GMT+offset formats. The behavior of the `V`, `VV` and `VVV` format/parse symbols has also changed. If you wish to specify timezones by name, you should always use the location-based names, such as "America/New_York" using the format/parse symbol `VV`. For more information, see "Format specification for the TimeFormat functions" in *Developing Apama Applications*.
  - When parsing time strings, trailing characters in the string after matching the whole pattern are ignored.
  - When parsing time strings, separator characters (such as a hyphen (-) or others which do not have a specific meaning) will match any such string of those characters of any length (at least one). This means that the pattern "yyyy-MM-dd" will now match "2016//11/15".
  - Some locale settings have fixes that may change the returned value for the "week of year" (that is, `w` and `W`).

- The *`chainInstanceId`* argument of the `createChain` method must now specify a unique identifier. Thus, an error is now thrown if a chain with a duplicate ID is to be created. See also "Creating dynamic chains from EPL" in *Connecting Apama Applications to External Components*.
- There is a change in functionality in the case of YAML configuration files. With previous versions, it was possible to specify, for example, `${PARENT_DIR}foo` on Windows. This was a valid specification as `PARENT_DIR` was expanded with a trailing slash. As of Apama 10.0, this is no longer valid. It is now required that you also specify the slash:

  ```
  ${PARENT_DIR}/foo
  ```

  This behavior is now in line with `PARENT_DIR` on Linux, and with `APAMA_HOME` and `APAMA_WORK` on Windows and Linux.

- The message metadata in connectivity plug-ins can now also store non-string values. For connectivity plug-ins in C++, these changes are not backwards compatible. See "Connectivity plug-ins enhancements in 10.0" on page 15 for more information.
- The `nullTerminated` configuration option of the String codec now defaults to `false`. This means that the String codec no longer adds a null-terminator in messages towards the transport when converting to a `byte[]` (Java) or `buffer_t` (C++) type. This change has been made to be more compatible with the common use case. To enable the old behavior (such as when using the Universal Messaging transport connectivity plug-in to connect to a legacy Apama application), set `nullTerminated` to `true` in the String codec configuration. See also "The String codec connectivity plug-in" in *Connecting Apama Applications to External Components*.
- When mapping `nEventProperties` to metadata, the Universal Messaging transport connectivity plug-in now uses the prefix `um.properties`. You can use the Mapper codec to enable the original behavior. See also "Connectivity plug-ins enhancements in 10.0" on page 15.
- The Mapper and Classifier codecs can now contain a period (.) for both payload and metadata rules. Custom codecs and transports may need changes to support this. See also "Connectivity plug-ins enhancements in 10.0" on page 15.
- The ConnectivityPlugins.mon file now also requires the new ConnectivityPluginsControl.mon file from the same directory. This is important if you inject files manually. If you are using Software AG Designer to develop applications that use any connectivity plug-in bundles, this does not matter to you as Software AG Designer automatically upgrades the set of .mon files that you are using.
- As of 10.0, `ifpresent` and `any` are reserved words in EPL. Projects that are using either of these as an identifier will fail in 10.0.

Removed and deprecated features in 10.0

The following Apama features are now deprecated or have been removed in Apama 10.0:

- The `subscribe()`, `unsubscribe()` and `sendToDES()` actions on the auto-generated EPL representations of Digital Event Services types are now deprecated. The channel name is now accessed via the static `CHANNEL` constant on the EPL type. For more information, see "Using Digital Event Services connectivity from EPL" in *Connecting Apama Applications to External Components*.
- For the Java and C++ connectivity plug-ins, the `LOGGER` field and use of the legacy constructor signatures that do not use `TransportConstructorParameters` or `CodecConstructorParameters` are now deprecated.

  For C++, the following macros are now deprecated:

  - `SAG_DECLARE_CONNECTIVITY_CODEC`

- `SAG_DECLARE_CONNECTIVITY_TRANSPORT`

  You should now use the new signature for the constructor, the new `logger` field, and the new macros as announced in "Connectivity API changes in 9.12.0.1" on page 32. For more information, see "Requirements of a plug-in class" in *Connecting Apama Applications to External Components*.

- The `UMStringCodec` is now deprecated. Use the new generic String codec instead. For detailed information, see "The String codec connectivity plug-in" in *Connecting Apama Applications to External Components*.
- For connectivity plug-ins, the `getMetadata()` function is now deprecated. Use the new `getMetadataMap()` function instead. See "Connectivity plug-ins enhancements in 10.0" on page 15 for more information.
- Support for the old APIs for writing EPL plug-ins in C and C++ is now deprecated and will be removed in a future release. EPL plug-ins should now be written using the new C++ API. See also "New API for writing EPL plug-ins in C++ in 10.0" on page 18.
- Dashboard support for applet deployment is no longer supported. The available deployment options are therefore Web Start, Local, or Display Server.
- Compiling with Visual Studio 2013 is no longer supported. See "Windows compiler update in 10.0" on page 23 for more information.
- The following information pertains to the Log File Manager plug-in (`LoggingManager`) which has been removed in 9.12, except for the associated bundle which was only removed in 10.0. If you still have the deprecated Logging Manager bundle in an Apama project, this results in an error indicating that the LoggingManager.bnd bundle cannot be found. To remove the Logging Manager bundle from an Apama project in Software AG Designer, proceed as follows:

  1. Select the project, invoke the context menu and choose **Properties**.
  2. In the properties dialog, select **Apama > MonitorScript Build Path**.
  3. Go to the **Bundles** tab.
  4. Select the **Cannot read bundle file** message and click **Remove**.
  5. Click **OK**.
  6. Click **Yes** to confirm the deletion of the bundle instance files.

3 What's New In Apama 9.12

Apama 9.12 is the successor of Apama 9.10. There is no version 9.11.

Apama 9.12 runs on the platforms listed in the *Supported Platforms* document for version 9.12. 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 9.12 includes new features, enhancements, and changes as described in the following topics.

Apama integration with Software AG Digital Event Services in 9.12

Apama now supports Software AG Digital Event Services, which is a messaging system for communicating between different Software AG products using events. Digital Event Services itself has event definitions of its own, called digital event types, that all participating products can have in common. For more information, see "The Digital Event Services Transport Connectivity Plug-in" in *Connecting Apama Applications to External Components*.

Using Software AG Designer, you can now add a new connectivity bundle named **Digital Event Services** to your projects. You use it to automatically generate the Apama event types from the Software AG Digital Event Services type definitions.

The new connectivity bundle includes the Digital Event Services connectivity plug-in. You use this plug-in to send and receive the generated Apama events. All of the configuration for this plug-in is driven by Software AG Designer, and you can edit the configuration by just changing a few properties.

For further information, see "Using the Digital Event Services connectivity bundle" in *Using Apama with Software AG Designer*.

Apama integration with Universal Messaging in 9.12

The Apama support for Universal Messaging has changed. A new connectivity plug-in is now available for connecting to Universal Messaging realms and channels. This provides for new features in mapping of message types on Universal Messaging.

Using Software AG Designer, you can add a new connectivity bundle named **Universal Messaging** to your projects. See "Adding the Universal Messaging connectivity plug-in to a project" in *Using Apama with Software AG Designer*. This bundle includes the Universal Messaging connectivity plug-in. All of the configuration for this plug-in is driven by Software AG Designer, and you can edit the configuration by just changing a few properties. For more information, see "The Universal Messaging Transport Connectivity Plug-in" and especially the topic "Configuring the Universal Messaging connectivity plug-in" in *Connecting Apama Applications to External Components*.

The new Universal Messaging connectivity plug-in replaces the old native Universal Messaging integration, which will be removed in a future release. It is recommended that you use the new connectivity plug-in instead.

New optional type in 9.12

EPL now supports `optional` types. An `optional` is a value that may be a value of some other specified EPL type, or empty and thus have no value. This is useful for mapping to `null` values in other languages such as Java, or for results or fields which may be missing a value. See "optional" in *Developing Apama Applications* for more information.

`optional` values can be extracted using the new `ifpresent` statement. This statement is used to check if one or more values are empty. It unpacks the values into new local variables and conditionally executes a block of code. See "Defining conditional logic with the ifpresent statement" and "The ifpresent statement" in *Developing Apama Applications* for more information.

Note: The `ifpresent` keyword can no longer be used as a valid identifier.

The connectivity plug-ins can now handle values of `optional` type. This means that an empty `data_t` or a `null` value in Java will translate to/from an empty optional. See also "Map contents used by the apama.eventMap host plug-in" in *Connecting Apama Applications to External Components*.

The Apama event parser libraries for Java and .NET have been improved to support events with `optional` fields. A new class `OptionalFieldType` (accessible via `FieldTypes.Optional`) has been added to represent the `optional` field type. For complete information about this class, refer to the description of

- `com.apama.event.parser` in the *API Reference for Java (Javadoc)*
- `Apama.Event.Parser Namespace` in the *API Reference for .NET*

Note: `optional` event field support will not be available in the C parser in `AP_EventParser` and `AP_EventWriter`.

New Microsoft Edge support in 9.12

Apama now supports the Microsoft Edge web browser.

Microsoft Edge does not support Java plug-ins and therefore does not support deployment of Apama dashboards as Java applets. If you need to deploy Apama dashboards as Java applets, use Microsoft Internet Explorer or Mozilla Firefox instead.

The supported versions of Microsoft Edge, Microsoft Internet Explorer and Mozilla Firefox are listed in the *Supported Platforms* document for version 9.12. This is available

from the following web page: "http://documentation.softwareag.com/apama/index.htm".

Connectivity plug-ins enhancements in 9.12

Apama 9.12 includes the following connectivity plug-ins enhancements:

- The `plugins` stanza that is used in the configuration file for the connectivity plug-ins has been renamed to `connectivityPlugins`. It is recommended that you start using the new name in your configuration files. See also "Overview of using connectivity plug-ins" in *Connecting Apama Applications to External Components*.
- The `--connectivityConfig` option which is available for the `correlator` tool has been renamed to `--config`. It is recommended that you start using the new name. In addition, it is now possible to specify multiple .yaml files containing connectivity plug-in configuration, which will be merged together. It is also possible now to specify a directory containing .yaml and .properties files. It is no longer possible to specify a YAML file that does not have the .yaml file extension; this will now result in an error. See also "Starting the correlator" in *Deploying and Managing Apama Applications*.
- A new `-D` option is available for the `correlator` tool, which specifies a value for a substitution property to be used by configuration files specified with the `--config` option of the `correlator` tool. See "Starting the correlator" in *Deploying and Managing Apama Applications*. Using this option is similar to specifying the substitution using a .properties file.
- Properties file substitution is now supported, allowing any user-supplied `${varname}` substitution variables to be replaced inside the configuration files for the connectivity plug-ins (.yaml files). See "Using properties files" in *Deploying and Managing Apama Applications*.
- Chains can now be defined under both the `startChains` and `dynamicChains` stanzas in the configuration file. Chains defined under `dynamicChains` are not instantiated at startup, but instead can be created dynamically from EPL. See also "Configuration file for connectivity plug-ins" and "Creating dynamic chains from EPL" in *Connecting Apama Applications to External Components*.

  Dynamic chains are also used by the Digital Event Services transport and the Universal Messaging transport.
- A new predefined Unit Test Harness codec is provided. The Unit Test Harness codec and the new Null Transport transport make it easy to send test messages into a connectivity chain and/or to write out received messages to a text file, without the need to write any EPL in the correlator, which is very useful for writing unit tests for connectivity plug-ins, and especially codecs. For more information, see "The Unit Test Harness codec connectivity plug-in" in *Connecting Apama Applications to External Components*.
- A new predefined Diagnostic codec is provided, which can be used to diagnose issues with connectivity plug-ins. It logs the events that go through a connectivity

chain in either direction. For more information, see "The Diagnostic codec connectivity plug-in" in *Connecting Apama Applications to External Components*.

- The behavior of the Mapper codec has changed as follows: A `mapFrom` rule where the source field does not exist now uses the default value if the `defaultValue` exists or if a subsequent `mapFrom` rule exists for the same destination field. An error is no longer raised in this case.

  For example, let us assume that there is no field `b` in the metadata and the configuration looks as follows:

  ```
  mapperCodec:
    SomeType:
      towardsHost:
      mapFrom:
        - payload.a: metadata.b
      defaultValue:
        - payload.a: "someDefaultValue"
  ```

  In the previous version, an error was raised with the message that `b` does not exist. As of 9.12, the default value for the payload field `a` (that is, `someDefaultValue`) is used rather than raising an error. However, if the destination field (which is `payload.a` in the above example) is also not defined with `defaultValue`, then an error is raised.

  In addition, a new boolean `allowMissing` is available. By default, this is set to `false`. When set to `true`, an error is not raised when a `defaultValue` has not been set and a source field is missing. `allowMissing` needs to be defined at the same level as the event types. For more information, see "The Mapper codec connectivity plug-in" in *Connecting Apama Applications to External Components*.

- The behavior of the `allowMissing` configuration property in the `apama.eventMap` host plug-in has changed. Previously only missing fields were set to their default values. As of this version, fields with empty values (`null` in Java) are also set to their default values. Similarly, empty values in nested events, elements in sequences and key/value pairs in dictionaries are also set to their default values. See also "Translating EPL events using the apama.eventMap host plug-in" in *Connecting Apama Applications to External Components*.

- New configuration properties `suppressLoopback`, `description` and `remoteAddress` are now available for the `apama.eventMap` and `apama.eventString` host plug-ins. For more information, see "Host plug-ins and configuration" in *Connecting Apama Applications to External Components*.

- You can now use the `jvmOptions` key in the configuration file to specify JVM options which the correlator is to pass to the embedded JVM. See also "Specifying JVM options" in *Deploying and Managing Apama Applications*.

- If you are building C++ plug-ins, you now need to link against the `apclient` library (instead of `apconnectivity`). See also "Building C++ plug-ins" in *Connecting Apama Applications to External Components*.

Connectivity API changes in 9.12.0.1

Version 9.12.0.1 introduces a new signature for the constructor of a C++ or Java connectivity plug-in, which passes in a `TransportConstructorParameters` or `CodecConstructorParameters` object instead of a list of individual parameters.

For Java, there is a new `logger` field that uses the Simple Logging Facade for Java (SLF4J) instead of the Log4J `LOGGER` available in earlier versions.

For C++, there is a new `logger` field to match the Java field, but the behavior is identical to `LOGGER` although we recommend to use the new name where possible for consistency.

In C++ only, there are the following new macros which must be used for classes with the new 9.12.0.1 constructor:

- `SAG_DECLARE_CONNECTIVITY_CODEC_CLASS` (instead of `SAG_DECLARE_CONNECTIVITY_CODEC`)
- `SAG_DECLARE_CONNECTIVITY_TRANSPORT_CLASS` (instead of `SAG_DECLARE_CONNECTIVITY_TRANSPORT`)

The previous legacy constructors, macros and logger are still available but will be deprecated in a future release. Therefore, we recommend migrating to the new ones when possible, especially for any newly created plug-ins.

To migrate from the old C++ transport constructor and macro to the new one, change the following:

```
class MyPlugin: public AbstractSimpleTransport
{
public:
  MyPlugin(const std::string &name, const std::string &chainId,
   const map_t &config)
    : AbstractSimpleTransport(name, chainId, config)
  {
    // do stuff with config here
    // use "LOGGER" for logging
  }
...
}
SAG_DECLARE_CONNECTIVITY_TRANSPORT(MyPlugin)
```

to:

```
class MyPlugin: public AbstractSimpleTransport
{
public:
  MyPlugin(const TransportConstructorParameters &params)
    : AbstractSimpleTransport(params)
  {
    // do stuff with params.getConfig() (or the config class member) here
    // recommended to use "logger" for logging instead of "LOGGER"
  }
...
}
// use new 9.12.0.1 macro for declaring the class
SAG_DECLARE_CONNECTIVITY_TRANSPORT_CLASS(MyPlugin)
```

The same applies to the C++ codecs, just replace "transport" with "codec".

To migrate from the old Java transport constructor and logger to the new one, change the following:

```
public MyTransport(Map<String, Object> config, String chainId,
  org.apache.log4j.Logger log4jLogger) throws Exception {
    super(config, chainId, log4jLogger);
    // do stuff with params.getConfig() (or the config class member) here
    LOGGER.info("Transport was created correctly with config: "+config);
...
```

to:

```
public MyTransport(org.slf4j.Logger logger,
  TransportConstructorParameters params) throws Exception {
    super(logger, params);
    logger.info("Transport was created correctly with config: "
      +params.getConfig());
...
```

The same applies to the Java codecs, just replace "transport" with "codec".

Be sure to change all uses of the legacy "LOGGER" to the new "logger" in your Java plug-ins. Most of the logger methods are identical (except that the new logger does not permit logging an exception without an accompanying string message).

See "Requirements of a plug-in class" in *Connecting Apama Applications to External Components* for more information about plug-in constructors.

Apama enhancements in Software AG Designer 9.12

Apama 9.12 includes the following enhancements in Software AG Designer:

- When you create an adapter, the instance name that Software AG Designer offers by default now
  - includes the adapter type,
  - no longer includes a space (thus, it is no longer required to escape the name in various places), and
  - the number that is provided with the name is incremented depending on the adapter type.

  For example, when you create a File adapter, the default instance name is now "FileAdapter1", and when you create a Web Services Client adapter next, the default instance name is now "WebServicesClientAdapter1". The instance names are no longer "instance 1" for the first adapter and "instance 2" for the second adapter.

  It is recommended that you specify your own instance names instead of just accepting the default names. An instance name should clearly identify the purpose of the adapter. For example, if you use the correlator-integrated adapter for JMS to connect to a customer tracking system, you should use a name such as "CustomerTracking" (and not just "JMS").

When you specify your own instance name, it is no longer possible to specify spaces within the name.

- Software AG Designer now supports adding multiple User Connectivity configuration instances (each represented by a directory of configuration files) to your project. The instances can contain multiple .yaml and .properties files. You can selectively enable or disable individual User Connectivity instances from the correlator launch configuration dialog. See "Creating Apama projects" in *Using Apama with Software AG Designer* for more information about adding User Connectivity support to a project.
- The **Connectivity Plug-ins Support** option has been removed from under **Startup options** in the Correlator Configurations dialog, and the **Connectivity** tab has been added.
- In the dialog for configuring correlators in a launch configuration, the **Xconfig** text box has been removed and replaced with a text box called **Configuration**. The **Configuration** text box accepts the path of a configuration file in either YAML (preferred) or the deprecated TXT configuration file format. This configuration can be used for a variety of purposes such as configuring per-package EPL logging settings. See "Configuring the correlator" in *Deploying and Managing Apama Applications*.

Query enhancements in 9.12

Apama 9.12 includes the following query enhancements:

- A query may now also use the result of an action call on the event as the key for a query. For further information, see "Defining actions as query keys" in *Developing Apama Applications*.
- Coassignment in query files can now use the `as` operator. For example:

```
using com.apama.aggregates.last;
query Example {
    find Event as e select last(e.value) as lastValue without Stop as s {
    }
}
```

Software AG Designer has been updated to use the `as` operator when constructing queries graphically. The current `:` assignment operator is still valid for backwards compatibility. For further information, see "Defining event patterns" in *Developing Apama Applications*.

EPL enhancements in 9.12

Apama 9.12 includes the following EPL enhancements:

- The following new `string` methods are available to search and replace strings using regular expressions:

- `matches(string regex)`
- `search(string regex)`
- `replace(string regex, string replacement)`

For further information, see "Additional methods which use regular expressions" in *Developing Apama Applications*. You can find this in the "EPL Reference", in the description of the `string` type.

Apama currently uses ICU 3.8.1 to implement the regular expressions.

- Event listeners (`on` statements) and stream listeners (`from` statements) can now coassign matching events to an implicitly declared variable using the `as` operator. This variable is only in scope in the processing block of the listener and has no effect on clashing local or global variables. For example:

```
// Variable newTick is not yet defined
on StockTick(*,*) as newTick {
    // The variable newTick is only in scope in this block
}
```

The current `:` assignment operator is still available for backwards compatibility. See "Specifying the on statement" and "Using output from streams" in *Developing Apama Applications* for further information.

- Data views now support the `decimal` field type. Decimals can thus be displayed in the Scenario Browser and used in dashboards. An event with a `decimal` field type can now be sent from a dashboard.
- Decimals are now also supported in C, C++ and Java plug-ins, in the MemoryStore and in the distributed MemoryStore.

  In C and C++, the decimal value is passed as an `AP_decimal` which is really a `uint64` in the IEEE 754-2008 decimal format. We suggest using the Intel Decimal Floating Point library v2.0U1 (see "https://software.intel.com/en-us/articles/intel-decimal-floating-point-math-library") if the actual decimal value needs to be used.

  In Java, both `BigDecimal` (for backwards compatibility) and `Number` are now supported types for decimal in a plug-in API. For `Number`, it is either a `BigDecimal` type for a value, or a `Double` type for NaN/infinity. NaN/infinity are still not supported when using the `BigDecimal` type, and will throw an exception that kills the monitor instance if not caught.
- It is now possible to generate ApamaDoc from an Ant script on UNIX. For further information, see "Generating ApamaDoc from an Ant script" in *Developing Apama Applications*.
- A new `OutOfBandConnections` event has been added to the `com.apama.oob` package, which provides a mechanism to synchronously get the currently connected senders and receivers. For further information, see "Out of band connection notifications" in *Developing Apama Applications*.
- It is now possible to declare static actions inside an event type. For further information, see "Defining static actions" in *Developing Apama Applications*.

- The `if` statement no longer requires the `then` keyword. It can still be specified optionally (for backwards compatibility), but you can now write a shorter form. For example:

  ```
  if seq.size() > 0 { print "First element is " + seq[0]; }
  ```

  See also "Defining conditional logic with the if statement" and "The if statement" in *Developing Apama Applications*.

Correlator utility enhancements in 9.12

Apama 9.12 includes the following correlator utility enhancements:

- It is now possible to configure persistence using a YAML configuration file that is specified with the new `--config` option when starting the correlator. For detailed information, see "Configuring persistence in a YAML configuration file" in *Deploying and Managing Apama Applications*.
- It is now possible to deploy Apama applications using YAML configuration files. This is useful for Docker containers or other minimal environments where only part of an Apama installation is available or it is not practical to run Java tools to perform the injections. For detailed information, see "Deploying Apama applications with a YAML configuration file" in *Deploying and Managing Apama Applications*.
- A new option `--pidfile` *`file`* is available for the `correlator`, `iaf`, `dashboard_server` and `display_server` tools. It specifies the name of the file that contains the process ID. For more information, see:
  - "Starting the correlator" in *Deploying and Managing Apama Applications*
  - "IAF command line options" in *Connecting Apama Applications to External Components*
  - "Command line options for the Data Server and Display Server" in *Building and Using Dashboards*
- A new request type `toStringQueues`, which can be issued with the `-r` (or `--dorequest`) option, is available for the `engine_management` tool. It outputs the current contents of all input and output queues within the running correlator. This can be helpful for identifying slow senders and receivers, and potential causes (such as very large events or excessive flow). See also "Shutting down and managing components" in *Deploying and Managing Apama Applications*.
- A new request type `getEventTypes`, which can be issued with the `-r` (or `--dorequest`) option, is available for the `iaf_management` tool. It returns a string representation of the event types known to the running IAF. The output is equivalent to that of the `iaf -e` *`config.xml`* command. The `getEventTypes` request type, however, does not require a configuration file as input. See also "Shutting down and managing components" in *Deploying and Managing Apama Applications*.

Dashboard enhancements in 9.12

Apama 9.12 includes the following dashboard enhancements:

- A new dashboard data table "Query instance table" is now available in the Dashboard Builder. Similar to scenario instance table, the new query instance table contains values of all input parameters for instances of an Apama query. You can define commands in the Dashboard Builder to create, edit and delete an instance of a query using the following commands:
  - Create query instance
  - Edit query instance
  - Delete query instance
  - Delete all instances of a query
- A new Tab Control object is now available on the **Controls** tab of the Dashboard Builder's Object Palette. This object provides an alternative to using a Tab Panel ini file.
- The REST API now supports the following new URI `GET` request, both for XML and JSON: `/info/stats`. This displays the statistics of instance/trend table queries to Dashboard Data Servers and Display Servers. In addition, `/connections` now also shows the connected correlators as well as any configured named Dashboard Data Servers and Display Servers. See also "Managing and Monitoring over REST" in *Deploying and Managing Apama Applications*.

PySys enhancements in 9.12

Apama 9.12 includes the following PySys enhancements:

- Apama 9.12 ships a new release of PySys, version 1.2.0. This new release includes some usability enhancements such as more informative messages when an assertion fails. For full details, see the PySys release notes.
- The PySys Apama `Correlator` class has a new method called `initialize()` which can be used to initialize the correlator with all the files making up an Apama application (such as .mon, .qry and .evt) by specifying just the path of the Software AG Designer project's .deploy file, or a .txt file or directory, without the need to list all the files making up the application individually. See the Apama PySys samples and Pydoc for more information and an example of how to use this.

Command Central support in 9.12

Apama 9.12 now supports the following in Command Central:

- Installing Apama using Command Central web user interface.
- Creating and deleting instances using the web user interface.
- Configuring correlator engine connect options. You can use this configuration to connect a source correlator (the sender) to a target correlator (the receiver). The target correlator will receive events from the specified channels of the source correlator. For more information, see "Configuring pipelining with engine_connect" in *Deploying and Managing Apama Applications*.
- Configuring correlator initialization. You can use this configuration to set an Apama application to initialize the correlator on startup.
- Configuring and deploying projects using the Digital Event Services connectivity plug-in.
- Monitoring KPIs for EPL applications.
- Monitoring KPIs for dashboard servers.

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

Miscellaneous enhancements and changes in 9.12

Apama 9.12 includes the following miscellaneous enhancements and changes:

- Apama 9.12 incorporates the ICU (International Components for Unicode) Timezone Data update 2016f, which is the most recent update at the time of release. This will update timezone data used by the correlator and Time Format EPL plug-in.
- The BigMemory Max driver now supports supplying converters per table to handle non-Apama managed data. The following new properties, which can be set in the spring.xml configuration file, are available for this purpose:
  - `converterConfig.default`
  - `converterConfig.byTable`

  For more information, see "BigMemory Max driver specific details" in *Developing Apama Applications*.
- In Apama versions 9.10 and prior, Universal Messaging channel names were escaped so that only a small subset of the ASCII characters were used in channel names in Universal Messaging. As of Apama 9.12, the new boolean configuration property `um.channels.escaped` is available which specifies whether channel names are escaped or not. To maintain backwards compatibility, this property is currently set to `true`, but this may change in future releases. If you set `um.channels.escaped` to `false`, Apama passes channel names directly to Universal Messaging without escaping. For more information, see "Defining Universal Messaging properties for Apama applications" in *Connecting Apama Applications to External Components*.

> **Note:** The information in the above topic, which also explains the new `um.channels.escaped` property, is deprecated. It applies to the old Universal Messaging integration which will be removed in a future release. It is recommended that you use the new Universal Messaging connectivity plug-in.

When the slash (/) and backslash (\) characters are not escaped, they can now be used to create nested channels. However, you must take care not to use both slash and backslash characters as path separators within the same application as this will result in undefined behavior. See also "Considerations for using Universal Messaging channels" in *Connecting Apama Applications to External Components*.

- The Docker samples have been updated to support the currently supported Docker and Docker Compose versions.

Platform changes in 9.12

Apama 9.12 runs on the platforms listed in the *Supported Platforms* document for version 9.12. This is available from the following web page: "http://documentation.softwareag.com/apama/index.htm".

Check the above mentioned *Supported Platforms* document for details about the newer platform versions supported by Apama 9.12. Due to these upgrades, the following versions are no longer supported:

- Software AG Terracotta BigMemory 4.3.2 (replaced by support for a more recent version).
- Software AG Universal Messaging 9.10 (replaced by support for a more recent version).
- Software AG Designer 9.10 (replaced by support for a more recent version).
- Software AG Common Runtime 9.10 (replaced by support for a more recent version).
- Oracle Java 8 update 71 (replaced by support for a more recent version).
- Google Chrome 40 (replaced by support for a more recent version).
- Firefox Mozilla 38 (replaced by support for a more recent version).
- SUSE Linux Enterprise Server 11 (replaced by support for a more recent version).
- Oracle WebLogic Server 12c 12.1.1 (replaced by support for a more recent version - only supported as a JMS provider).
- Apache Ant 1.7.1 (replaced by support for a more recent version).
- Docker 1.6.0 (replaced by support for a more recent version).
- Docker Compose 1.3.0 (replaced by support for a more recent version).

Miscellaneous changes in 9.12 affecting backwards compatibility

The following changes in Apama 9.12 affect backwards compatibility to previous Apama versions:

- Due to tightening of the `canParse` and `parse` methods on the EPL types of `integer`, `float` and `decimal`, some previously allowed behavior is now invalid. These EPL types now only allow trailing and leading whitespace, excluding special characters on the `float` and `decimal` types such as "d".

  As of this version, `integer` parsing will only accept integers and whitespace. Any other characters are invalid, which also means that whole numbers such as `"1.0"` and `"2.0"` will not parse. The following examples illustrate the new behavior for the `canParse` and `parse` methods:

Now false	Was previously parsed to
" "	0
"2.2"	2
"2e3"	2
"2 2"	2
"2garbage"	2

  `float` and `decimal` parsing remains unchanged, except that hexadecimal integers will no longer parse. The following example illustrates the new behavior for the `canParse` and `parse` methods:

Now false	Was previously parsed to
0x1d	0

- In previous releases, any invalid keyword arguments passed to methods in the Apama PySys extension classes were silently ignored with no error. For example, the following did not produce an error despite being incorrect:

  ```
  correlator.start(foobar='x')
  ```

  As of this release, an exception will be thrown if an invalid argument is specified. Thus, test cases that have errors in them are now easy to notice.

Removed and deprecated features in 9.12

The following Apama features are now deprecated or have been removed in Apama 9.12:

- Event Modeler is now deprecated. Support for creating and deploying scenarios will be removed in a future release. It is recommended that you use EPL or queries to build new Apama applications.
- The EngineManagement API for C is now deprecated. It is recommended that you use the EngineManagement API for C++ instead.
- The old native Universal Messaging integration is now deprecated and will be removed in a future release. It is recommended that you use the new Universal Messaging connectivity plug-in instead. See also "Apama integration with Universal Messaging in 9.12" on page 28.
- The `-r` (or `--rnames`) and `-UMconfig` (or `--umConfigFile`) options of the `correlator` tool, which pertain to the old native Universal Messaging integration, are now deprecated. Use the `--config` option together with one or more .yaml files and (typically) also .properties files instead.
- The `um.install.dir` property, which is used by the old native Universal Messaging integration to set the location of the Universal Messaging libraries in the UM-config.properties file, is now deprecated. If Universal Messaging has been installed in the same Software AG installation directory as Apama, Apama will now locate the Universal Messaging installation automatically at ${APAMA_HOME}/../UniversalMessaging and will load the Universal Messaging libraries from the appropriate subdirectory. See also "Setting up Universal Messaging for use by Apama" in *Connecting Apama Applications to External Components*.
- The `-Xconfig` option of the `correlator` tool, which specifies a special configuration file (see "Using the Apama component extended configuration file" in *Deploying and Managing Apama Applications*), is now deprecated in favor of YAML configuration via `--config`. For detailed information on configuration using YAML, see "Configuring the correlator" and the description of the `--config` option in "Starting the correlator", both in *Deploying and Managing Apama Applications*.
- As of this version, the deprecated `profiling` request can no longer be used with the `-r` option of the `engine_management` tool. Instead, you now have to use the `cpuProfile` request, which provides the same functionality. For further information, see "Using the CPU profiler" in *Deploying and Managing Apama Applications*. See also the announcement in "Removed and deprecated features in 9.10" on page 56.
- When using the `engine_management` tool with the `-r` option, it is no longer possible to specify a single string, enclosed in quotation marks, which delimits multiple arguments by spaces. As of this version, it is only possible to specify the arguments as shown in the following example:

```
engine_management -r cpuProfile frequency
```

Make sure that multiple arguments are no longer enclosed in quotation marks. Quotation marks are now only used for arguments that contain spaces, such as a file name. See also the announcement in "Correlator utility enhancements in 9.9" on page 61.

- As of this version, the deprecated Log File Manager plug-in (`LoggingManager`) can no longer be used. Instead, you now have to use the built-in logging functionality. For further information, see "Logging and printing" in *Developing Apama Applications*. See also the announcement in "Removed and deprecated features in 9.9" on page 65.
- As of this version, the deprecated Enterprise Management and Monitoring (EMM) and sentinel agent features can no longer be used. It is recommended that you now use Command Central for all deployment tasks. For detailed information, see "Deploying Apama Components with Command Central" in *Deploying and Managing Apama Applications*. See also the announcement in "Removed and deprecated features in 9.10" on page 56.
- The apamadoc.xml and apamadoc.bat files have been removed from the utilities directory of the Apama installation. To generate ApamaDoc without a user interface on both Windows and UNIX, you now have to use the `generate-apamadoc` macro from the apama-macros.xml Ant script, which is located in the etc directory of your Apama installation. For further information, see "Generating ApamaDoc from an Ant script" in *Developing Apama Applications*.
- As of this version, the Apama EPL Visualization tool for Software AG Designer is no longer installed with Apama. The tool is now available for download from the Apama area of the Software AG TECHcommunity website at "http://techcommunity.softwareag.com".

4 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 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 51 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 51 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 54 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`*`,`*`rank`*`)`

  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 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.

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 54), 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.

5 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_studion.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.

6 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:

- "Implementing a custom Java mapper"
- "Using custom EL mapping extensions"
- "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.
- The behavior of escaping in mapping literal expressions in the correlator JMS and Web Services Client adapter has been changed. Previously, literal expressions in mapping were evaluated as is without any escaping. Now a slash (\) is used as the escape character. Now a double quote (") is escaped as `\"`, a single quote (') is escaped as `\'` and a slash (\) is escaped as `\\`. Quotes only need to be escaped if a string value is enclosed in the same type of quote. So the literal expression "a quote \' is string" was previously evaluated as "a quote \' is string", but it is now evaluated as "a quote ' is string".
- Several performance improvements have been made to the `Parse()` and Text (formatting) implementation of the .NET event parser. This has resulted in a small change of behavior in `Event.FieldsMap`, which previously returned a `SortedDictionary` with a deterministic key iteration order (based on the natural ordering of the keys) but now returns an unordered dictionary with non-deterministic iteration order. For rare cases where deterministic ordering is required, users should wrap the `FieldsMap` result in a new `SortedDictionary` instance before iterating over it.
- The default consistency mode for BigMemory configurations is now `EVENTUAL` (the BigMemory default) rather than `STRONG`, and `synchronousWrites` has been disabled. Existing configurations will not be updated automatically, and users are still able to enable `STRONG` consistency if needed, but due to its superior performance, `EVENTUAL` is recommended when using Apama with the latest BigMemory release.

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 70.

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

7 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" in *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 *Connecting Apama Applications to External Components*.

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 87.

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 83 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" in *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 IAF adapters to use Universal Messaging" 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 Universal Messaging 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 83.

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 EL 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.

  The change in behavior of `engine_connect` switching to persistent connections and other changes in Apama messaging means that two connections for the same component and channel but using different hostnames for the connections (for example, some combination of full hostname, short hostname or IP address, or

multiple different addresses or hostnames) will create multiple connections, potentially duplicating events. Apama recommends ensuring all correlators are named consistently. In default (not parallel) mode, multiple connections using the same hostname will be flattened to a single connection, avoiding duplicate events.

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`

- The `DataViewService_Impl_Dict` monitor now uses the built-in EPL `log` statement instead of having its own custom logger, and the `com.apama.dataview.SetLogLevel` event has been removed. The log level for this monitor can now be changed as described in "Setting logging attributes for packages, monitors and events" in *Deploying and Managing Apama Applications*.

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 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;
  ```

8 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 to a deployment script" 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*.

9 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 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 "Exception handling" in *Developing Apama Applications*.

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 *Deploying and Managing 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 "Writing EPL 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 "Generating ApamaDoc from an Ant script" in *Developing Apama Applications*.

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").