What's New in Apama

5.2.0

August 2014

Document ID: PAM-What's_New_in_Apama-5.2.0-20140808@233876

Table of Contents

Preface

About this documentation

What's New in Apama describes the changes introduced with the Apama 5.2 release as well as earlier 5.0 releases.

Preface

Contacting customer support

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

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

Preface

Chapter 1: What's New in Apama 5.2

Apama 5.2 runs on the platforms listed in the Apama 5.2 Support Matrix available from a link on the following web page: empower.softwareag.com. Be sure to consult that page 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 Apama Studio*.

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.

What's New in Apama 5.2

New feature for addressing contexts with channels in 5.2

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

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

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

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

EPL

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

Correlator utilities

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

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

Apama client APIs

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

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

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

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

Correlator plug-in APIs

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

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

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

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

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

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

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

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

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

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

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

Adapter configurations

- When specifying connections to correlators, you can specify the optional `parallelConnectionsLimit` 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 *Developing Adapters*.

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 11.

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 for details.
- In correlator plug-ins, use the new methods for subscribing to receive events sent on particular channels and for sending events to particular channels. These methods are provided by:
  - C++: `AP_CorrelatorInterface`
  - Java: `com.apama.epl.plugin.Correlator`

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

- In `engine_connect` command lines, specify the `-m parallel` or `--mode parallel` option. In parallel mode, for each specified channel, Apama will create a separate connection between the sending and target correlators.
- In event files to be sent by the `engine_send` utility, specify channels. For details, see "Event association with a channel" at the end of the *Event Correlator Utilities Reference* section of *Deploying and Managing Apama Applications*.

- In the parts of your application that receive output from the `engine_receive` utility, make any changes needed to accommodate the new channel specification associated with events.

What's New in Apama 5.2

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 messaging 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" and "Configuring adapters to use UM" in *Deploying and Managing Apama Applications*.

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

Details for using UM in an Apama application are in *Deploying and Managing Apama Applications*, *Using Universal Messaging*.

What's New in Apama 5.2

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 7.

What's New in Apama 5.2

Enhancements to Apama adapters in 5.2

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

For details, see "Using custom mapping extensions" in *Deploying and Managing Apama Applications*.

What's New in Apama 5.2

Changes to correlator utilities in 5.2

Apama 5.2 enhances and changes correlator utilities as follows:

- `engine_connect`

  The `engine_connect` utility has a new parallel mode (`-m parallel` or `--mode parallel`) that uses a connection per channel between correlators and delivers events to the target correlator on the

channel the events were sent from in the source correlator. Previous behavior of `engine_connect` is preserved in legacy mode (`-m legacy` or `--mode legacy`), which uses one connection between two correlators and delivers all events to the default channel.

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

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

- `engine_debug`

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

- `engine_inspect`

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

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

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

- `engine_management`

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

- `engine_receive`

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

- `engine_send`

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

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

- `engine_watch`

  This utility now provides the following information:

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

- `extract_replay_log`

  When you run this script on a correlator input log you can specify the `-c` or `--correlator` option. This option specifies that the script that `extract_replay_log` generates should include the command line for starting a correlator. When you run the generated script, the correlator will be started with all of the command line options needed to replay the input log. While this option is not new in this release, it was previously undocumented.

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

What's New in Apama 5.2

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.

What's New in Apama 5.2

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.

What's New in Apama 5.2

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 .

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

Oracle

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

DB2

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

What's New in Apama 5.2

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`

What's New in Apama 5.2

Miscellaneous enhancements and changes in 5.2

Apama 5.2 includes the following enhancements:

- In an IAF adapter configuration file, the new default behavior is that all sinks defined for an Apama event correlator configuration now receive the events generated by the IAF Semantic Mapper. To change the default behavior, add the `sendEvents="false"` attribute to each `<sink>` element that represents a component that should not receive events. In previous releases, the default behavior was that only the first sink would receive events.
- Several changes have been made to the `EngineClient`, `EventService` and `ScenarioService` layers in Java and .NET to make it easier to use event and property change listeners safely without risking thread deadlock situations. As a result of these changes, Apama client developers might notice that engine client property change listeners now fire asynchronously (on a separate dedicated thread), which is a slight change in behavior. See the release notes for details.
- When using EDA events in Apama applications, non-EDA namespaces are now allowed. Also, namespaces that contain colons (`:`) can now be converted to Apama package names.
- In Dashboard Builder, when you define the Send event command, you can optionally specify the channel on which to send the event.
- The new Correlator Time Format dashboard function lets you convert a correlator timestamp to epoch time or to a date/time format you specify.
- The BigMemory MemoryStore driver now sets the Java `type` of each search attribute, which makes it easier to use the search attributes from other products. For example, Software AG's Presto can extract the `ehcache.xml` configuration from a running BigMemory instance.

- The distributed MemoryStore `RowValue` API class has a number of usability improvements in its API, including easier to use set and get methods. (The old-style set methods have been deprecated.) There is also a new `RowValueHelper` class that allows more efficient allocation of `RowValue` objects, and simplifies getting, setting and iterating over the contents of a `RowValue` object.
- Apama 5.2 incorporates ICU (International Components for Unicode) Timezone Data update 2014e, which is the most recent update at the time of release. This will update timezone data used by the correlator and TimeFormat correlator plug-in.
- C++ correlator plug-ins can be declared as nonblocking to prevent creation of unneeded processing threads. If a method in a nonblocking plug-in might block you can override the nonblocking designation for that method.
- The C++ plug-in API method for sending an event has a new overloading, which you should now use:

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

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

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

What's New in Apama 5.2

Chapter 2: What's New in Apama 5.1.1

Apama 5.1.1 includes the following enhancements:

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

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

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

  For details, see *Exporting a Launch Configuration* in *Using Apama Studio*.
- 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 in EPL*.
- 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*.

Chapter 3: 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 please consult the appropriate What's New and Migration information available with the intermediate versions. For further assistance please contact the 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 Apama Studio*.

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 in EPL*.

What's New in Apama 5.1

New EPL exception handling mechanism

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

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

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

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

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

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

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

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

What's New in Apama 5.1

New support for managing and monitoring with REST APIs

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

Apama components expose several URIs which can be used to either monitor or manage different parts of the system. Generic management URIs are exposed by most Apama components, while other URIs are exposed only by specific types of components. Most URIs are purely for informational purposes and will only respond to HTTP `GET` requests and interacting with them will not change the state of the component. However, some URIs allow the state of the correlator to be modified. For example, to set a component's log level, the `/logLevel` URI accepts an HTTP `PUT` request containing an XML document that specifies the log level.

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

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="/resources/transform.xslt"?>
<map name="apama-response">
   <prop name="numConsumers">0</prop>
```

```
   <prop name="numOutEventsQueued">0</prop>
   <prop name="numOutEventsUnAcked">0</prop>
   <prop name="numOutEventsSent">0</prop>
   <prop name="uptime">67657</prop>
   <prop name="numMonitors">0</prop>
   <prop name="numProcesses">0</prop>
   <prop name="numJavaApplications">0</prop>
   <prop name="numListeners">0</prop>
   <prop name="numEventTypes">0</prop>
   <prop name="numQueuedFastTrack">0</prop>
   <prop name="numQueuedInput">0</prop>
   <prop name="numReceived">0</prop>
   <prop name="numFastTracked">0</prop>
   <prop name="numEmits">0</prop>
   <prop name="numProcessed">0</prop>
   <prop name="numSubListeners">0</prop>
   <prop name="numContexts">1</prop>
   <prop name="virtualMemorySize">262540</prop>
   <prop name="numSnapshots">0</prop>
   <prop name="numInputQueuedInput">0</prop>
   <prop name="mostBackedUpInputContext"><none></prop>
   <prop name="mostBackedUpICQueueSize">0</prop>
   <prop name="mostBackedUpICLatency">0.0</prop>
</map>
```

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

What's New in Apama 5.1

New support for creating correlator plugins using Java

Custom correlator plugins can now be written in Java as well as in C and C++. When a Java plugin is injected to the correlator, it is analyzed and any suitable method in the plugin can be called from applications running in the correlator. Correlator plugins 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 Plugin written in Java" in *Using Apama Studio*.
- To write a correlator plugin in Java from scratch, see "Writing Correlator Plugins in Java" in *Writing Correlator Plugins*.

What's New in Apama 5.1

New headless mode generation of ApamaDoc

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

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

What's New in Apama 5.1

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.

What's New in Apama 5.1

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.

What's New in Apama 5.1

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`

What's New in Apama 5.1

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 messaging 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 *What's New in Apama*.)

What's New in Apama 5.1

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

What's New in Apama 5.1

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).

What's New in Apama 5.1

Chapter 4: What's New in Apama 5.0.1

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

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

Linux runtime performance enhancement

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

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

The interpreted runtime compiles EPL into bytecode whereas the compiled runtime compiles EPL into native code that is directly executed by the CPU. For many applications, the compiled runtime provides significantly faster performance than the interpreted runtime. Applications that perform dense numerical calculations are most likely to have the greatest performance improvement when the correlator uses the compiled runtime. Applications that spend most of their time managing listeners and searching for matches among listeners typically show a smaller performance improvement.

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

- The interpreted runtime allows for the profiler and debugger to be switched on during the execution of EPL. The compiled runtime does not permit this. For example, you cannot switch on the profiler or debugger in the middle of a loop.
- The amount of stack space available is different for the two runtimes. This means that recursive functions run out of stack space at a different level of recursion on the two runtimes.

What's New in Apama 5.0.1

Changes to the Apama Database Connector

LogCommands and LoqQueries

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

What's New in Apama 5.0.1

Integrating JMon applications with Corticon

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

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

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

What's New in Apama 5.0.1

Correlator-integrated messaging adapter for JMS bundle name changes

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

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

What's New in Apama 5.0.1

Improved profiling analysis

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

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

For more information on this feature, see "Profiling EPL Applications" in *Using Apama Studio*.

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

What's New in Apama 5.0.1

5.0 features now documented

Documentation has been added for the following Apama 5.0 features:

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

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

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

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

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

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

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

  - `getHostname()` - Returns the host name of the host the correlator is running on. The host name is dependent on the environment's name resolution configuration, and the name can be used only if the name resolution is correctly configured. The name is the same as that logged in the correlator log file, for example, `dev3.acme.com`.
  - `getComponentPort()` - Returns the port the correlator is running on.
  - `getComponentPhysicalId()` - Returns the physical ID of the correlator.
  - `getComponentLogicalId()` - Returns the logical ID of the correlator.

- `getComponentName()` - Returns the name that is used to identify the correlator. You can set this name by specifying the `-N` correlator command line flag (or by means of the `extraArgs` attribute in the ANT macros). The default name of the correlator is `correlator`.

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

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

What's New in Apama 5.0.1

Specifying Dashboard Builder options in Apama Studio

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

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

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

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

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

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

What's New in Apama 5.0.1

Chapter 5: What's New in Apama 5.0

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

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

Changes to EPL features in Apama 5.0

In Apama 5.0, EPL includes the following new features and changes:

- Backwards incompatible change to Time Format correlator plug-in

  The Time Format correlator plug-in `parse()` functions now return `NaN` when they cannot parse the specified string. In previous releases, these functions returned `-1` when they could not parse the specified string. This applies to the `parse()`, `parseTime()`, `parseUTC()`, `parseTimeUTC()`, `parseWithTimeZone()`, and `parseTimeWithTimeZone()` functions.

- Backwards incompatible change to non-parseable objects

  In previous releases, if you called the `parse()` method on one of the following objects, the code compiled but failed at runtime. With this release, such code does not compile.

  - An event that contains an unparseable field
  - A sequence that contains unparseable items
  - A dictionary whose key or item type is unparseable
- Change in `enqueue...to` behavior

  The following code now causes the correlator to terminate the monitor instance because it enqueues an event to a default context value rather than an actual context. In previous releases, this code did nothing silently.

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

  If you enqueue an event to a sequence of contexts and one of the contexts has the default value then the correlator terminates the monitor instance.
- Stream queries
  - A `stream` type can now contain any type of item. Previously, a `stream` type was restricted to containing `boolean`, `float`, `integer`, `string`, or `event` type items.
  - You can now specify a `having` clause to filter the items produced from the stream query's projection.
  - Projections are now categorized as one of simple istream, simple rstream or aggregate. Simple istream and rstream projections do not use aggregates and cannot be grouped; the `rstream` keyword is not useful in conjunction with an aggregate projection, and this usage has been withdrawn.
- EPL now supports the `decimal` primitive type. When perfect accuracy in working with decimal fractions is a requirement, use the `decimal` type in place of the `float` type. When you are not working with decimal values then using `float` types is the right choice. For example, it makes sense to use `decimal` types to compute 3.9% interest on $127,729.23 if it is important that the answer be exact. Conversely, there is no reason to use a `decimal` type to determine the area of circle, for example, because pi is not a decimal fraction. When extremely small floating point variations are acceptable, you might choose to use the `float` type to obtain better performance.

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

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

  Correlator plug-ins, Event Modeler, DataViews, and dashboards do not support use of the `decimal` type.
- New `string` methods
  - The new `string.intern()` method is an optimization that can reduce memory use. This method takes no arguments, returns the interned version of the string that it is called on, and marks the string as being interned, which means that the correlator uses the same string object for every subsequent occasion that the same string is parsed.

  - The new `string.replaceAll(string1, string2)` method makes a copy of the string that the method is called on, replaces instances of *`string1`* in the copy with instances of *`string2`*, and returns the updated copy.
  - The new `string.split(string)` method splits the string that is the argument at occurences of the string that the method is called on and returns a sequence that contains the new strings.
  - The new `string.tokenize(string)` method divides the `string` argument into tokens where each token is separated from its neighbors by one or more delimiters in the string that the method is called on. The `tokenize()` method returns a sequence of strings.
- It is now possible to use any comparable type as the key in a dictionary. The comparable types are:
  - `boolean`
  - `decimal`
  - `float`
  - `integer`
  - `string`
  - `context`
  - `dictionary` if it contains items that are a comparable type
  - `event` if it contains only comparable types
  - `location`
  - `sequence` if it contains items that are a comparable type
- New `dictionary` methods:
  - The `dictionary.getOr(key, alternative)` method either retrieves an existing item by its key or returns the specified alternative.
  - The `dictionary.getOrDefault(key)` method either retrieves an existing item by its key or returns a default instance of the dictionary's item type if the dictionary does not contain the specified key.
  - The `getOrAdd(key, alternative)` method either retrieves an existing item by its key or if the dictionary does not already contain an item with the specified key then it adds the specified key to the dictionary with the specified value and then returns that value.
  - The `getOrAddDefault(key)` method either retrieves an existing item by its key or if the dictionary does not already contain an item with the specified key then it adds the specified key to the dictionary with a default-constructed value and returns that value.

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

  Before Apama 5.0, the `dictionary.getOr()` method was called `dictionary.getDefault()`. The `getDefault()` method remains only for backwards compatibility and should not be used. It is deprecated and will be removed in a future release. Use the `dictionary.getOr()` method instead.
- The new `sequence.setCapacity(integer)` method sets the amount of memory initially allocated for the sequence. Note that this does not limit the amount of memory the sequence can use. By default, as you add more elements to a sequence, the correlator allocates more memory.

Calling `sequence.setCapacity()` can improve performance because it removes the need to add more memory each time you add an element to the sequence. For example, consider a sequence that will contain at least 1000 elements. A call to `setCapacity(1000)` removes the need to allocate additional memory unless more than 1000 elements are added. A call to this method does not change the behavior of your code.

- The new `context.isPublic()` method returns a Boolean value that indicates whether the context is public.
- New `chunk` methods
  - `empty()` – returns true if the chunk is empty. This lets you distinguish between a chunk that contains a default initialization value and a chunk that has been populated by a correlator plug-in.
  - `getOwner()` – returns a string that contains the name of the correlator plug-in that the chunk belongs to. This method returns an empty string if the chunk is empty.
- Keywords
  - `persistent` is now a keyword. It is an error to use it as an identifier unless you prefix it with a hash symbol (#).
  - The `streamsource` keyword is reserved for future use. It is an error to use `streamsource` as an identifier unless you prefix it with a hash symbol (#).
  - The `generates` keyword is no longer reserved for future use. You can use `generates` as an identifier.
- The `onBeginRecovery()` and `onConcludeRecovery()` actions are now handled by the correlator in the same way as the other special actions, `onload()`, `ondie()`, and `onunload()`. That is, the correlator ensures that any `onBeginRecovery()` and `onConcludeRecovery()` actions do not take arguments and do not return values. It is a compile-time error if you define one of these actions with any other signature. Also, you cannot use `onBeginRecovery` or `onConcludeRecovery` as the name for any of the following:
  - Global variable (monitor scope)
  - Event field
  - Aggregate function field
- When you expose a MemoryStore table as a DataView you can now specify the display name and/or the description for the exposed DataView. Previously, the MemoryStore always used a default display name and description.
- The names `open`, `close` and `reset` are no longer reserved for future use. A custom aggregate function can have members with these names.
- The following sample event interfaces and monitors, some of which are specific to capital market solutions, have been removed from Apama 5.0:
  - `DatabaseSupport.mon`
  - `MultiLegOrderManagerSupport.mon`
  - `OrderManagerSupport.mon`
  - `ScenarioOrderService.mon`
  - `SimpleExchangeSimulator.mon`

  - SimpleNonPersistentDatabase.mon
  - SimplePriceGenerator.mon
  - TickManagerSupport.mon

- Payload Extraction correlator plug-in is deprecated. All Apama adapter products now use dictionary-format payloads rather than string-format payloads. The dictionary-format payload is considerably more efficient in almost all cases. While string-format payloads are still accepted in Apama 5.0, this support, including the Payload Extraction correlator plug-in, is now deprecated and will be removed in a future release of the Apama platform. It is recommended that all third-party adapters are migrated to dictionary-format payloads as soon as possible.
- The EPL `call` statement was deprecated in previous releases and has been removed in this release. To invoke an action, see "Defining actions" in *Developing Apama Applications with EPL*.

What's New in Apama 5.0

The com.apama namespace is restricted

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

What's New in Apama 5.0

New Apama client API support for decimal type

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

Apama Client SDK for Java

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

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

Apama .NET Client SDK

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

can be retrieved from the property `StringValue`. The `ToString` of this class will stringify the Dot Net Decimal value whenever it is not null, otherwise it will stringify the `float` representation. There are also helper properties available such as `IsNaN` and `IsInfinity`.

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

Apama C/C++ Client SDK

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

What's New in Apama 5.0

New ADBC adapter features

In Apama 5.0, the ADBC adapter has been enhanced in a variety of areas. For complete up-to-date information, see "Using the Apama Database Connection" in *Deploying and Managing Apama Applications*.

General API improvements

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

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

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

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

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

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

- `Connection.openDatabaseFull`

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

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

Token parameter added

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

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

The following new actions have been added for use with the `token` parameter:

- `Query.getQueryResultToken`
- `Query.getBatchDoneToken`

New playback action

The following action has been added for running playback queries:

- `Query.setRunUntilTime`

A SQL statement can be specified for store actions

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

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

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

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

New FixedSizeInsertBuffers property

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

Database connections can be opened in read-only mode

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

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

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

New open "shared" actions

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

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

New reconnection capability

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

- `action setReconnectPolicy`
- `action setReconnectTimeout`

What's New in Apama 5.0

Packaged Progress database drivers

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

- DB2
- Microsoft SQL Server
- OpenEdge
- Oracle

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

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

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

`<apama_install_dir>\doc\db_drivers\jdbc\books.pdf`

`<apama_install_dir>\doc\db_drivers\odbc\books.pdf.`

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

What's New in Apama 5.0

New Web Services client adapter features

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

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

For complete information on the Web Services client adapter, see "The Apama Web Services Client Adapter" in *Deploying and Managing Apama Applications*.

What's New in Apama 5.0

New correlator-integrated messaging for JMS features

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

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

- Apama provides new sample applications that illustrate the use of correlator-integrated messaging for JMS. The *apama_dir*\samples\correlator_jms directory contains the following sample applications:
  - simple-send-receive - This application demonstrates simple sending and receiving. It sends a sample event to a JMS queue or topic as a JMS TextMessage using the automatically configured default sender and receives the message using a statically-configured receiver.
  - dynamic-event-api - This application demonstrates how to use the event API to dynamically add and remove JMS senders and receivers. In addition, it shows how to monitor senders and receivers for errors and availability.
  - flow-control - This application demonstrates how to use the event API to avoid sending events faster than JMS can cope with and a separate demonstration of how to avoid receiving messages from JMS faster than the EPL application can cope with.

For information on these features, see "correlator-integrated messaging for JMS" in *Deploying and Managing Apama Applications* and the ApamaDoc, which is available at *apama_install_doc*\doc\ApamaDoc\index.html

What's New in Apama 5.0

Event parser usability/type-safety improvements

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

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

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

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

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

- The event parser library now explicitly prevents setting or getting null field values when using the new getField and setField methods. This simplifies coding and ensures you get a helpful message instead of a NullPointerException when getting a field value that was not assigned.
- The event parser library supports method chaining in setField to allow "fluent" coding style, allows fully initializing an event instance without needing to assign it to a temporary variable
- Added FieldType.newField(), DictionaryFieldType.type(), and SequenceFieldType.type() helper methods. These methods avoid the need to duplicate the generic type parameters on the RHS (as

well as the LHS) when constructing fields and types. Using `fieldType.newField(name)` saves time and effort as compared to using `new Field<type parameter>(name, fieldType)`.

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

What's New in Apama 5.0

Correlator Deployment Packages

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

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

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

  For more information on `engine_package`, see "Event Correlator Utilities Reference" in *Deploying and Managing Apama Applications*.

- The `engine_inject` utility has a new `-c` (`--cdp`) option that is used to inject CDP files.

  For more information on `engine_inject`, see "Event Correlator Utilities Reference" in *Deploying and Managing Apama Applications*.

- The Apama C/C++ API has the following new methods for injecting CDP files:

  - `virtual void injectCDP(const AP_uint8* cdpbytes, AP_uint32 size, const char *filename=NULL) = 0;`
  - `virtual const char* const* injectCDPWithWarnings(const AP_uint8* cdpbytes, AP_uint32 size) = 0;`
  - `virtual const char* const* injectCDPWithWarningsFilename(const AP_uint8* cdpbytes, AP_uint32 size, const char *filename) = 0;`

  For more information on the C++/C API, see "The Client Software Development Kits for C++ and Java" in *Developing Clients*.

- The Apama client API for Java has the following new methods for injecting CDP files:

  - `void injectCDP(byte[] cdpBytes)`
  - `void injectCDP(byte[] cdpBytes, java.lang.String filename)`
  - `void injectCDPsFromFile(java.util.List<java.lang.String> filenames)`

  For more information on these methods, see "The JavaBeans API" in *Developing Clients*.

- The Apama client API for .NET has methods equivalent to the Apama client API for Java methods for injecting CDP files. For more information on the .NET client

API, see the Apama .NET documentation help file located at *install_dir*\doc\dotNet \engine_client_dotnet5.0.chm

- In Apama Studio, you can use CDP files by adding them to your projects in the following ways:
  - As external dependencies to the MonitorScript Build Path.
  - Directly including them in a project, for example, by dragging the CDP file from Windows Explorer and dropping it onto the project in Apama Studio's Project Explorer view.

  For more information on adding resources to Apama projects, see "Working with Projects" in *Using Apama Studio*.
- Apama Studio provides a new feature to export EPL files as a CDP file. For information on the Export as Correlator Deployment Package wizard, see "Working with Projects" in *Using Apama Studio*.

What's New in Apama 5.0

New correlator startup option

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

What's New in Apama 5.0

Changes to engine_send

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

- The

  `--doNotBatch | -d`

  This argument has been removed from `engine_send`. Now all event files will be automatically batched by `engine_send`.
- BATCH tags

  `BATCH` tags without timestamps are now ignored by `engine_send`. Batching will be performed by `engine_send` and the client library. `BATCH` tags with timestamps still behave the same. All events in the previous batch will be flushed followed by a sleep until the next batch time.
- Multi-line, ml-style comments removed

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

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

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

What's New in Apama 5.0

Deprecated serialize and deserialize operations have been removed

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

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

What's New in Apama 5.0

Deprecated high availability components removed

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

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

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

What's New in Apama 5.0

Changes in Apama Studio

Apama profiler

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

- The profiler no longer uses the Eclipse Test and Performance Tools Platform (TPTP).
- Now you can launch and profile the entire launch configuration in single operation.
- A launch shortcut available for the Apama project.

- You can take snapshots of data and compare the information in the snapshots with incoming, current data.
- More than one correlator of the project can be profiled in one shot.
- The new profiler is more lightweight.
- Columns view can be filtered, ordered, and rearranged.
- View filters are more generic.
- Differences between snapshot and incoming data are more cleanly displayed.

Event definitions from XML and XSD files

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

Bundles removed

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

- DatabaseSupport
- FinanceSupport
- SimpleExchangeSimulator

What's New in Apama 5.0

Changes related to dashboards

Tomcat no longer bundled with Apama

The Tomcat application server and Tomcat-specific features have been removed from Apama. Progress Control Tower now bundles Tomcat and not JBoss. The Deploy/Publish wizard lets you locally or remotely deploy to Control Tower in either its bundled Tomcat or WebLogic application server. To use the WebSphere application server, you must deploy the dashboard manually.

Named Data Server support

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

Dashboard deployment changes

The Deploy/Publish wizard has been streamlined and simplified. It includes support for the Tomcat application server included with Progress Control Tower. Support has been removed for PCT 1.x-

style input to the wizard and Deployment Configuration editor. For more information, see Using the Deployment Configuration editor in Preparing Dashboards for Deployment in *Using Apama Studio*. See also Deploying Dashboards in *Deploying and Managing Apama Applications*.

Dashboard message localization support

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

Dashboard attachment and command changes

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

New dashboards panels options

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

Enhanced tree control

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

Fx objects removed

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

Data Server status table change

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

New and changed dashboard functions

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

- Concatenate Columns
- Delta And Rate Rows
- Ensure Columns
- Ensure Timestamp Column
- Format Table Columns
- Group By Time
- Group By Time And Unique Values
- Group By Unique Values

- Join Outer
- Rename Columns
- Table Contains Values
- Validate Substitutions

New Custom command

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

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

In the Command Value: field, enter the string:

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

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

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

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

New and changed visualization objects

The following visualization objects have been added or changed:

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

the user enters a blank string, the `actionCommand` will not execute and the `invalidInputVarToSet` and `invalidInputMsgVarToSet` will be updated to indicate an invalid entry.

- Slider control: A new property, `axisDirection`, allows you to set the axis direction of the scale to the following: Bottom to Top (vertical orientation with the minimum on the bottom and the maximum on top), Left to Right (horizontal orientation with the minimum on the left and the maximum on the right), Top to Bottom (vertical orientation with the minimum on the top and the maximum on the bottom), and Right to Left (horizontal orientation with the minimum on the right and the maximum on the left). Another new property, `fgColor`, sets the tick mark color. You can either select a color or accept the default color. The `fgColor` is only applied when the control is enabled, so tick marks on sliders in the main Builder window will not use it (preview your display to see the `fgColor` applied). This change does not apply to Display Server (thin client) deployments, whose sliders do not have tick marks.
- Date chooser control: This is enhanced to support three new properties: `fgColor` (the font color of the text entry area; default is black), `validColor` (the font color to use while editing in the text entry area if the entered value is using the correct format; default is green), and `invalidColor` (the font color used during and after editing in the text entry area if the entered value is not using the correct format; default is red). `validColor` and `invalidColor` are not supported in Display Server (thin client) deployments.
- Trend graph: A new property, `historyOnlyFlag`, controls whether only historical data is included. See the *Dashboard Property Reference* for more information.

What's New in Apama 5.0

Event Modeler standard block changes

In previous releases, Event Modeler included a number of standard blocks related to capital market applications. These blocks are no longer provided with Event Modeler. Instead, they are available with the Apama Capital Markets Solution Foundation. The blocks that have been moved are listed below:

- Basket Calculator
- EWMA Calculator
- MACD Calculator
- Market Depth
- Multi-Leg Order Manager
- OBV Calculator
- Order Flow
- Order Manager
- P&L Calculator
- Position Calculator
- RSI Calculator
- Volume Distributor
- VWAP Calculator

Also, the Database Storage and Database Retrieval blocks have been removed. In place of these two blocks, use the ADBC Storage and ADBC Retrieval blocks, which were added in a previous release.

What's New in Apama 5.0