Advanced Message Processing System (AMPS) Evaluation Guide

Advanced Message Processing System (AMPS) Evaluation Guide

4.3

Publication date Oct 29, 2015

Table of Contents

Chapter 1. Introduction to 60East Technologies AMPS

Thank you for choosing the Advanced Message Processing System (AMPS™) from 60East Technologies™. AMPS is more than a publish and subscribe system. It is a feature-rich platform that enables you to easily build data intensive applications that provide previously unattainable low latency and high performance. AMPS combines a set of capabilities that cut across traditional component divisions. 60East designed the capabilities based on the needs of some of the most demanding data-intensive applications on the planet, and engineered the capabilities to work together seamlessly and provide the kind of performance and latency that those applications demand.

1.1. Product Overview

AMPS, the Advanced Message Processing System, is built around an incredibly fast messaging engine that provides traditional publish-subscribe messaging and a wide array of advanced messaging features. AMPS combines the capabilities necessary for scalable high-throughput, low-latency messaging in real-time deployments such as in financial services. AMPS goes beyond basic messaging to include advanced features such as high availability, historical replay, aggregation and analytics, content filtering and continuous query, last value caching, and more.

Furthermore, AMPS is designed and engineered specifically for next generation computing environments. The architecture, design and implementation of AMPS allows the exploitation of parallelism inherent in emerging multi-socket, multi-core commodity systems and the low-latency, high-bandwidth of 10Gb Ethernet and faster networks. AMPS is designed to detect and take advantage of the capabilities of the hardware of the system on which it runs.

AMPS does more than just route and deliver messages. AMPS was designed to lower the latency in real-world messaging deployments by focusing on the entire lifetime of a message from the message's origin to the time at which a subscriber takes action on the message. AMPS considers the full message lifetime, rather than just the "in flight" time, and allows you to optimize your applications to conserve network bandwidth and subscriber CPU utilization -- typically the first elements of a system to reach the saturation point in real messaging systems.

AMPS offers both topic and content based subscription semantics, which makes it different than most other messaging platforms. Some of the highlights of AMPS include:

- Topic and content based publish and subscribe
- Client development kits for popular programming languages such as Java, C#, C++, C and Python
- Built in support for FIX, NVFIX, JSON, BSON and XML messages. AMPS also supports uninterpreted binary messages, and allows you to create composite message types from existing message types.
- State-of-the-World queries
- Historical State-of-the-World queries

- Easy to use command interface
- Full PERL compatible regular expression matching
- Content filters with SQL92 **WHERE** clause semantics
- Built-in latency statistics and client status monitoring
- Advanced subscription management, including delta publish and subscriptions and out-of-focus notifications
- Basic CEP capabilities for real-time computation and analysis
- Aggregation within topics and joins between topics, including joins between different message types
- Replication for high availability
- Fully queryable transaction log
- Message replay functionality
- Extensibility API for adding message types, transports, authentication, and entitlement functionality

1.2. Software Requirements

AMPS is supported on the following platforms:

- Linux 64-bit (2.6 kernel or later) on x86 compatible processors

While 2.6 is the minimum kernel version supported, AMPS will select the most efficient mechanisms available to it and thus reaps greater benefit from more recent kernel and CPU versions.

1.3. Document Conventions

This manual is an introduction to the 60East Technologies AMPS product. It assumes that you have a working knowledge of Linux, and uses the following conventions.

Table 1.1. Documentation Conventions

Construct	Usage
text	standard document text
`code`	inline code fragment
variable	variables within commands or configuration
	usage tip or extra information

Construct	Usage
⚠	usage warning
`required`	required parameters in parameter tables
optional	optional parameters in parameter tables

Additionally, here are the constructs used for displaying content filters, XML, code, command line, and script fragments.

```
(expr1 = 1) OR (expr2 = 2) OR (expr3 = 3) OR (expr4 = 4) OR (expr5 = 5) OR (expr6 = 6) OR (expr7 = 7) OR (expr8 = 8)
```

Command lines will be formatted as in the following example:

```
find . -name *.java
```

1.4. Obtaining Support

For an outline of your specific support policies, please see your 60East Technologies License Agreement. Support contracts can be purchased through your 60East Technologies account representative.

Support Steps

You can save time if you complete the following steps before you contact 60East Technologies Support:

1. Check the documentation. The problem may already be solved and documented in the *User's Guide* or reference guide for the product. 60East Technologies also provides answers to frequently asked support questions on the support web site at http://support.crankuptheamps.com.

2. Isolate the problem.

   If you require Support Services, please isolate the problem to the smallest test case possible. Capture erroneous output into a text file along with the commands used to generate the errors.

3. Collect your information.

   - Your product version number.
   - Your operating system and its kernel version number.
   - The expected behavior, observed behavior and all input used to reproduce the problem.
   - Submit your request.

- If you have a minidump file, be sure to include that in your email to `crash@crankuptheamps.com`.

The AMPS version number used when reporting your product version number follows a format listed below. The version number is composed of the following:

```
MAJOR.MINOR.MAINTENANCE.HOTFIX.TIMESTAMP.TAG
```

Each AMPS version number component has the following breakdown:

Table 1.2. Version Number Components

Component	Description
`MAJOR`	Increments when there are changes in functionality, file formats, configs, or deprecated functionality.
`MINOR`	Ticks when new functionality is added.
`MAINTENANCE`	Increments with standard bug fixing, maintenance, small features and enhancements.
`HOTFIX`	A release for a critical defect impacting a customer. A hotfix release is designed to be 100% compatible with the release it fixes (that is, a release with same MAJOR.MINOR.MAINTENANCE version)
`TIMESTAMP`	Proprietary build timestamp.
`TAG`	Identifier that corresponds to precise code used in the release.

Contacting 60East Technologies Support

Please contact 60East Technologies Support Services according to the terms of your 60East Technologies License Agreement.

Support is offered through the United States:

Toll-free:	(888) 206-1365
International:	(702) 979-1323
FAX:	(888) 216-8502
Web:	http://www.crankuptheamps.com
E-Mail:	`sales@crankuptheamps.com`
Support:	`support@crankuptheamps.com`

Chapter 2. Getting Started

This chapter is for users who are new to AMPS and want to get up and running on a simple instance of AMPS. This chapter will walk new users through the file structure of an AMPS installation, configuring a simple AMPS instance and running the demonstration tools provided as part of the distribution to show how a simple publisher can send messages to AMPS.

2.1. Installing AMPS

To install AMPS, unpack the distribution for your platform where you want the binaries and libraries to be stored. For the remainder of this guide, the installation directory will be referred to as `$AMPSDIR` as if an environment variable with that name was set to the correct path.

Within `$AMPSDIR` the following sub-directories listed in Table 2.1.

Table 2.1. AMPS Distribution Directories

Directory	Description
api	Include files for modules that work directly with the AMPS server binary
bin	AMPS engine binaries and utilities
docs	Documentation
lib	Library dependencies
sdk	Include files for the AMPS extension API

AMPS client libraries are available as a separate download from the AMPS web site. See the AMPS developer page at http://www.crankuptheamps.com/developer to download the latest libraries.

2.2. Starting AMPS

The AMPS Engine binary is named `ampServer` and is found in `$AMPSDIR/bin`. Start the AMPS engine with a single command line argument that includes a valid path to an AMPS configuration file. For example, you can start AMPS with the demo configuration as follows:

```
$AMPSDIR/bin/ampServer $AMPSDIR/demos/amps_config.xml
```

AMPS uses the current working directory for storing files (logs and persistence) for any relative paths specified in the configuration. While this is important for real deployments, the demo configuration used in this chapter does not persist anything, so you can safely start AMPS from any working directory using this configuration.

On older processor architectures, `ampServer` will start the `ampServer-compat` binary. The `ampServer-compat` binary avoids using hardware instructions that are not available on these systems.

If your first start-up is successful, you should see AMPS display a simple message similar to the following to let you know that your instance has started correctly.

```
AMPS 4.0.0.0 - Copyright (c) 2006 - 2014 60East Technologies, Inc.
(Built: Nov 16 2014 13:53:41)

For all support questions: support@crankuptheamps.com
```

If you see this, congratulations! You have successfully cranked up the AMPS!

2.3. Admin View of the AMPS Server

When AMPS has been started correctly, you can get an indication if it is up or not by connecting to its admin port with a browser at `http://<host>:<port>` where <*host*> is the host the AMPS instance is running on and <*port*> is the administration port configured in the configuration file. When successful, a hierarchy of information regarding the instance will be displayed. If you've started AMPS using the sample configuration file, try connecting to `http://localhost:8085`. For more information on the monitoring capabilities, please see *AMPS Monitoring Reference Guide*, available from the 60East documentation site at `http://docs.crankuptheamps.com/`.

2.4. Interacting with AMPS Using Spark

AMPS provides the `spark` utility as a command line interface to interacting with an AMPS server. `spark` provides many of the capabilities of the AMPS client libraries through this interface. The utility lets you execute commands like `'subscribe'`, `'publish'`, `'sow'`, `'sow_and_subscribe'` and `'sow_delete'`.

You can read more about `spark` in the `spark` chapter of the AMPS User Guide. Other useful tools for troubleshooting AMPS are described in the *AMPS Utilities Guide*.

2.5. JSON Messages - A Quick Primer

AMPS includes support for FIX, NVFIX, XML, JSON and BSON messages, as well as the ability to develop custom message types and to send binary payloads. This section is going to focus on JSON as the primary message type.

JSON format is a simple, standardized message format. JSON has two basic constructs:

- key / value pairs
- arrays of values

JSON supports hierarchical construction: the value for a key can be a single value, an array of values, or another set of key/value pairs. For example, the following JSON message includes two nested sets of key value pairs. Notice that a key only needs to be unique within each set of values -- the `name` value for the ship does not conflict with the `name` value for the character.

```
{ "id" : 73, "character" :
            { "name" : "Han Solo",
              "occupation" : "smuggler",
              "ship" :
              { "name"  : "Millennium Falcon",
                "speed" : ".5 past light speed",
                "cargo" :
                [ "widgets", "baskets", "spice"]
              }
            }
}
```

Many AMPS applications use JSON as the payload. In addition, the `amps` protocol used by most AMPS applications represents commands to AMPS in a JSON-format header. For example, a publish command might look like:

```
{"c":"publish","t":"test-topic"}{ "id" : 1, "message" : "Hello,
 World!" }
```

The command to AMPS, using the `amps` protocol, is a JSON document which contains the header information for AMPS -- in this case, a `publish` to the topic `test-topic`. The header is followed by the message body, the payload of the command.

While the `amps` protocol is implemented as JSON, you can use any message type with the amps protocol: the header for the command will still be JSON, while the body can be in the message type of your choice, as in the sample below, which publishes to an XML topic:

```
{ "c":"publish","t":"xml-topic"}<example><id>1</id><message>Hello,
 world!</message></example>
```

The AMPS client libraries create and parse AMPS headers. For example, the `publish` method in the AMPS client libraries creates the appropriate header for a publish command based on the provided parameters.

Your applications use the `Message` and `Command` interfaces of the AMPS client libraries to work with the AMPS headers. There is no need for your application to parse or serialize the AMPS headers directly.

The AMPS client libraries handle creating and parsing AMPS headers. They do not parse or interpret the payload data on received `Message`, instead returning the payload as a string.

Chapter 3. Publish and Subscribe

AMPS is a publish and subscribe message delivery system, which routes messages from publishers to subscribers. “Pub/Sub” systems, as they are often called, are a key part of most enterprise message buses, where publishers broadcast messages without necessarily knowing all of the subscribers that will receive them. This decoupling of the publishers from the subscribers allows maximum flexibility when adding new data sources or consumers.

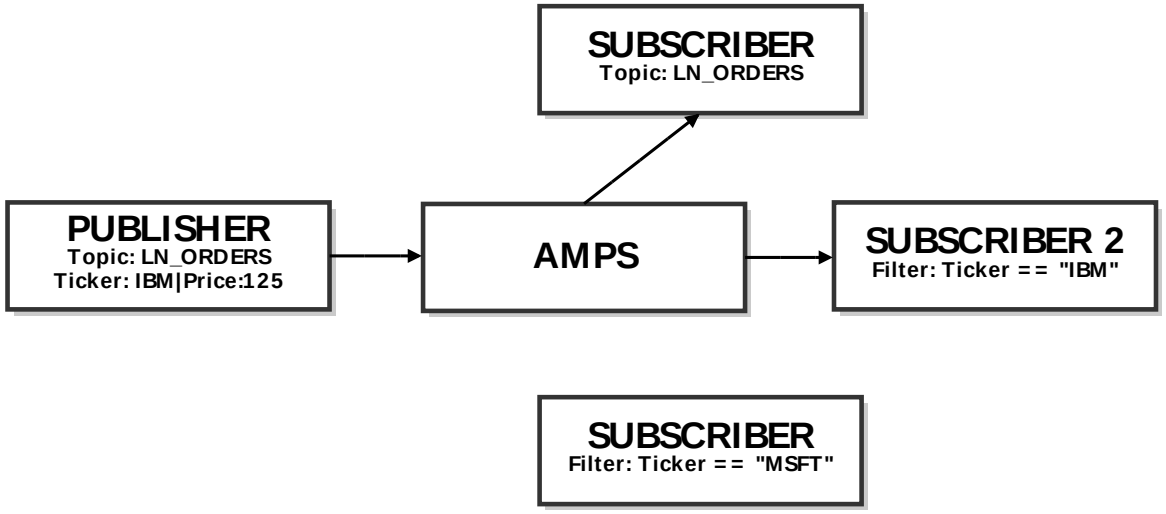

Figure 3.1. Publish and Subscribe

AMPS can route messages from publishers to subscribers using a topic identifier and/or content within the message's payload. For example, in Figure 3.1, there is a Publisher sending AMPS a message pertaining to the `LN_ORDERS` topic. The message being sent contains information on Ticker “IBM” with a Price of 125, both of these properties are contained within the message payload itself (i.e., the message content). AMPS routes the message to Subscriber 1 because it is subscribing to all messages on the `LN_ORDERS` topic. Similarly, AMPS routes the message to Subscriber 2 because it is subscribed to any messages having the Ticker equal to “IBM”. Subscriber 3 is looking for a different Ticker value and is not sent the message.

3.1. Topics

A topic is a string that is used to declare a subject of interest for purposes of routing messages between publishers and subscribers. Topic-based Publish and-Subscribe (e.g., Pub/Sub) is the simplest form of Pub/Sub filtering. All messages are published with a topic designation to the AMPS engine, and subscribers will receive messages for topics to which they have subscribed.

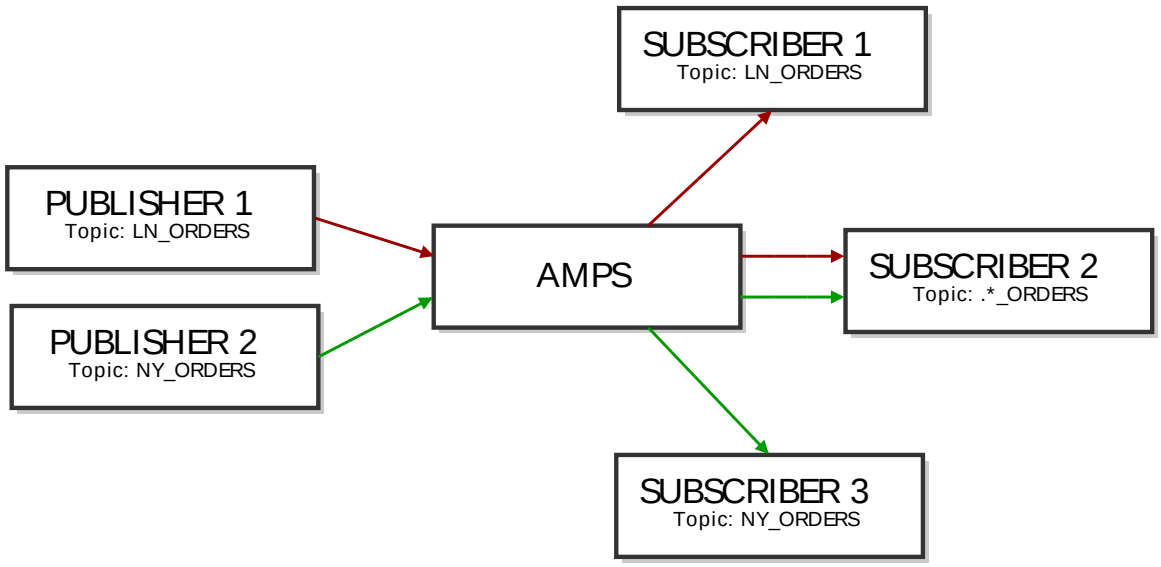

Figure 3.2. Topic Based Pub/Sub

For example, in Section 3.1, there are two publishers: Publisher 1 and Publisher 2 which publish to the topics `LN_ORDERS` and `NY_ORDERS`, respectively. Messages published to AMPS are filtered and routed to the subscribers of a respective topic. For example, Subscriber 1, which is subscribed to all messages for the `LN_ORDERS` topic will receive everything published by Publisher 1. Subscriber 2, which is subscribed to the regular expression topic `".*_ORDERS"` will receive all orders published by Publisher 1 and 2.

Regular expression matching makes it easy to create topic paths in AMPS. Some messaging systems require a specific delimiter for paths. AMPS allows you the flexibility to use any delimiter. However, 60East recommends using characters that do not have significance in regular expressions, such as forward slashes. For example, rather than using `northamerica.orders` as a path, use `northamerica/orders`.

Regular Expressions

With AMPS, a subscriber can use a regular expression to simultaneously subscribe to multiple topics that match the given pattern. This feature can be used to effectively subscribe to topics without knowing the topic names in advance. Note that the messages themselves have no notion of a topic pattern. The topic for a given message is unambiguously specified using a literal string. From the publisher's point of view, it is publishing a message to a topic; it is never publishing to a topic pattern.

Subscription topics are interpreted as regular expressions if they include special regular expression characters. Otherwise, they must be an exact match. Some examples of regular expressions within topics are included in Table 3.1.

Table 3.1. Topic Regular Expression Examples

Topic	Behavior
`^trade$`	matches only “trade”.
`^client.*`	matches “client”, “clients”, “client001”, etc.
`.*trade.*`	matches “NYSEtrades”, “ICEtrade”, etc.

For more information regarding the regular expression syntax supported within AMPS, please see the *Regular Expression* chapter in the *AMPS User Guide*.

AMPS can be configured to disallow regular expression topic matching for subscriptions. See the *AMPS Configuration Guide* for details.

3.2. Filtering Subscriptions By Content

One thing that differentiates AMPS from classic messaging systems is its ability to route messages based on message content. Instead of a publisher declaring metadata describing the message for downstream consumers, the publisher can simply publish the message content to AMPS and let AMPS examine the native message content to determine how best to deliver the message.

The ability to use content filters greatly reduces the problem of oversubscription that occurs when topics are the only facility for subscribing to message content. The topic space can be kept simple and content filters used to deliver only the desired messages. The topic space can reflect broad categories of messages and does not have to be polluted with metadata that is usually found in the content of the message. In addition, many of the advanced features of AMPS such as out-of-focus messaging, aggregation, views, and SOW topics rely on the ability to filter content.

Content-based messaging is somewhat analogous to database queries that include a `WHERE` clause. Topics can be considered tables into which rows are inserted (or updated). A subscription is similar to issuing a `SELECT` from the topic table with a `WHERE` clause to limit the rows which are returned. Topic-based messaging is analogous to a `SELECT` on a table with no limiting `WHERE` clause.

AMPS uses a combination of XPath-based identifiers and SQL-92 operators for content filtering. Some examples are shown below:

Example Filter for a JSON message

```
(/Order/Instrument/Symbol = 'IBM') AND
(/Order/Px >= 90.00 AND /Order/Px < 91.00)
```

Example Filter for an XML Message:

```
(/FIXML/Order/Instrmt/@Sym = ’IBM’) AND (/FIXML/Order/@Px
```

```
        >= 90.00 AND /FIXML/Order/@Px < 91.0)
```

Example Filter for a FIX Message:

```
/35 < 10 AND /34 = /9
```

For more information about how content is handled within AMPS, check out the *Content Filtering* chapter in the *AMPS User Guide*.

Unlike some other messaging systems, AMPS lets you use a relatively small set of topics to categorize messages at a high level and use content filters to retrieve specific data published to those topics. Examples of good, broad topic choices:

```
trades, positions, MarketData, Europe, alerts
```

This approach makes it easier to administer AMPS, easier for publishers to decide which topics to publish to, and easier for subscribers to be sure that they've subscribed to all relevant topics.

Replacing the Content Filter on a Subscription

AMPS allows you to replace the content filter on an existing subscription. When this happens, AMPS begins sending messages on the subscription that match the new filter. When an application needs to bring more messages into scope, this can be more efficient than creating another subscription.

For example, an application might start off with a filter such as the following

```
/region = 'WesternUS'
```

The application might then need to bring other regions into scope, for example:

```
/region IN ('WesternUS', 'Alaska', 'Hawaii')
```

Replacing a filter is an atomic operation. That is, the application is guaranteed not to miss messages that are in both the original and replacement filter, and is guaranteed to receive all messages for the new filter as of the point at which the replacement happens.

Chapter 4. State of the World (SOW)

One of the core features of AMPS is the ability to persist the most recent update for each message matching a topic. The State of the World can be thought of as a database where messages published to AMPS are filtered into topics, and where the topics store the latest update to a message. Since AMPS subscriptions are based on the combination of topics and filters, the State of the World (SOW) gives subscribers the ability to quickly resolve any differences between their data and updated data in the SOW by querying the current state of a topic, or a set of messages inside a topic.

4.1. How Does the State of the World Work?

Much like a relational database, AMPS SOW topics contain the ability to persist the most recent update for each message. AMPS identifies a message by using a unique key for the message. The SOW key for a message is similar to the primary key in a relational database: each value of the key is a unique message. The first time a message is received with a particular SOW key, AMPS adds the message to the SOW. Subsequent messages with the same SOW key value update the message.

AMPS assigns a SOW key based on the content of the message. The fields to use for the key are specified in the SOW topic definition, and consist of one or more XPath expressions. AMPS finds the specified fields in the message and computes a SOW key based on the name of the topic and the values in these fields.

The following diagrams demonstrate how the SOW works.

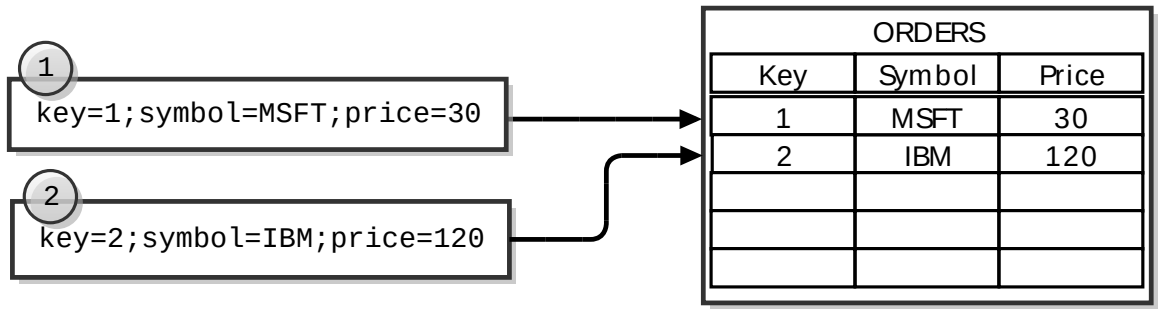

Figure 4.1. A SOW topic named `ORDERS` with a key definition of `/Key`

In Figure 4.1, two messages are published where neither of the messages have matching keys existing in the `ORDERS` topic, the messages are both inserted as new messages. Some time after these messages are processed, an update comes in for the `MSFT` order changing the price from 30 to 35. Since the `MSFT` order update has a key field of 1, this matches an existing record and overwrites the existing message containing the same key, as seen in Figure 4.2.

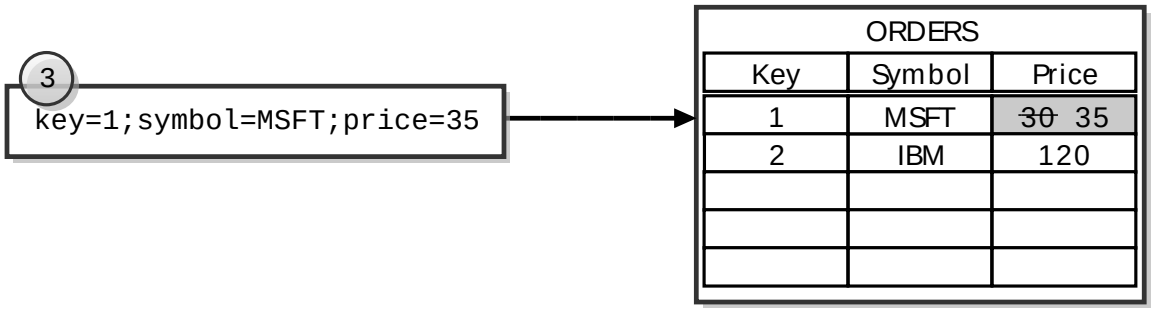

Figure 4.2. Updating the `MSFT` record by matching incoming message keys

By default, state of the world topics are *persistent*. For persistent topics, AMPS stores the contents of the state of the world in a dedicated file. This means that the total state of the world does not need to fit into memory, and that the contents of the state of the world database are maintained across server restarts. You can also define a *transient* state of the world topic, which does not store the contents of the SOW to a file.

The state of the world file is separate from the transaction log, and you do not need to configure a transaction log to use a SOW. When a transaction log is present that covers the SOW topic, on restart AMPS uses the transaction log to keep the SOW up to date. When the latest transaction in the SOW is more recent than the last transaction in the transaction log (for example, if the transaction log has been deleted), AMPS takes no action. If the transaction log has newer transactions than the SOW, AMPS replays those transactions into the SOW to bring the SOW file up to date. If the SOW file is missing, AMPS rebuilds the state of the world by replaying the transaction log from the beginning of the log.

When the state of the world is transient, AMPS does not store the state of the world across restarts. In this case, AMPS does not synchronize the state of the world with the transaction log when the server starts. Instead, AMPS tracks the state of the world for messages that occur while the server is running, without replaying previous messages into the SOW.

4.2. Queries

At any point in time, applications can issue SOW queries to retrieve all of the messages that match a given topic and content filter. When a query is executed, AMPS will test each message in the SOW against the content filter specified and all messages matching the filter will be returned to the client. The topic can be a literal topic name or a regular expression pattern. For more information on issuing queries, please see the *SOW Queries* chapter in the *AMPS User Guide*.

4.3. Configuration

Topics where SOW persistence is desired can be individually configured within the `SOW` section of the configuration file. Each topic will be defined with a `TopicDefinition` section enclosed within `SOW`.

The *AMPS Configuration Reference* contains a description of the attributes that can be configured per topic. `TopicMetaData` is a synonym for `SOW` provided for compatibility with previous versions of AMPS.

Table 4.1. SOW/TopicDefinition

Element	Description
`FileName`	The file where the State of the World data will be stored. This element is required for State of the World topics with a `Durability` of `persistent` (the default) because those topics are persisted to the filesystem. This is not required for State of the World topics with a durability of `transient`.
`MessageType`	Type of messages to be stored. To use AMPS generated SOW keys, the message type specified must support content filtering so that AMPS can determine the SOW key for the message. In this release, AMPS loads these message types that support content filtering: `fix`, `nvfix`, `json`, `bson`, and `xml`. The `binary` message type does not support content filtering. This message type does not support content filtering, so this message type can only be used for a SOW when publishers use explict keys.
`Topic`	The name of the SOW topic - all unique messages (see `Key`) on this topic will be stored in a topic-specific SOW database.
`Key`	Specifies an XPath within each message that AMPS will use to determine whether a message is unique. This element can be specified multiple times to create a composite key. A SOW topic can have either a key determined by AMPS, or publishers can provide the SOW key for a message with each message. 60East recommends having AMPS determine the key unless your application has specific needs that make this impractical. AMPS automatically creates a hash index for the SOW key.
`HashIndex`	AMPS provides the ability to do fast lookup for SOW records based on specific fields. When one or more `HashIndex` elements are provided, AMPS creates a hash index for the fields specified in the element. These indexes are created on startup, and are kept up to date as records are added, removed, and updated. The `HashIndex` element contains a `Key` element for each field in the hash index. AMPS uses a hash index when a query uses exact matching for all of the fields in the index. AMPS does not use hash indexes for range queries or regular expressions. AMPS automatically creates a hash index for the SOW key.

Element	Description
RecoveryPoint	For SOW topics that are covered by the transaction log, the point from which to recover the SOW if the SOW file is removed, or if the SOW topic has transient duration. This configuration item allows two values: • epoch recovers the SOW from the beginning of the transaction log • now recovers the SOW from the current point in the transaction log Defaults to epoch.
Index	AMPS supports the ability to precreate memo indexes for specific fields using the Index configuration option. When one or more Index elements are provided, AMPS creates memo indexes for any field specified in an Index element on startup, before a query that uses that field runs. Otherwise, AMPS indexes each field the first time a query uses the field. Adding one or more Index configurations to a TopicDefinition can improve retrieval performance the first time a query that contains the indexed fields runs for large SOW topics.
RecordSize	Size (in bytes) of a SOW record for this topic. Default: 512
InitialSize	Initial size (in records) of the SOW database file for this topic. Default: 2048
IncrementSize	Number of records to expand the SOW database (for this topic) by when more space is required. Default: 1000
Expiration	Time for how long a record should live in the SOW database for this topic. The expiration time is stored on each message, so changing the expiration time in the configuration file will not affect the expiration of messages currently in the SOW. AMPS accepts interval values for the Expiration, using the interval format described in the AMPS Configuration Guide section on units, or one of the following special values: • A value of disabled specifies that AMPS will not process SOW expiration for this topic, regardless of any expiration value set on the message. In this case, AMPS saves the expiration for the message, but does not process it. The value must be set to disabled (the default) if History is enabled for this topic.

Element	Description
	• A value of `enabled` specifies that AMPS will process SOW expiration for this topic, with no expiration set by default. Instead, AMPS uses the value set on the individual messages (with no expiration set for messages that do not contain an expiration value). Default: `disabled` (never expire)
`KeyDomain`	The seed value for `SowKeys` used within the topic. The default is the topic name, but it can be changed to a string value to unify `SowKey` values between different topics. For example, if your application has a `ShippingAddress` SOW and a `CreditRating` SOW that both use `/customerID` as the SOW key, you can use a `KeyDomain` to ensure that the generated `SowKey` for a given `/customerId` is identical for both SOW topics. This does not affect how AMPS processes the SOW topics, but can make correlating information from different SOW topics easier in your application. Default: the name of the SOW topic
`Durability`	Defines the data durability of a SOW topic. SOW databases listed as `persistent` are stored to the file system, and retain their data across instance restarts. Those listed as `transient` are not persisted to the file system, and are reset each time the AMPS instance restarts. Default: `persistent` Valid values: `persistent` or `transient` Synonyms: `Duration` is also accepted for this parameter for backward compatibility with configuration prior to 4.0.0.1
`History`	Enable historical query for this SOW. This element contains a `Window` and `Granularity` element. When the `History` element is present, historical query is enabled for this sow. Otherwise, AMPS does not enable historical query and does not store the historical state of the SOW. `Expiration` must be `disabled` when `History` is enabled.
`Window`	For a historical SOW, the length of time to store history. For example, when the value is `1w`, AMPS will store one week of history for this SOW. Used within the `History` element. Default: By default, AMPS does not expire historical SOW data.
`Granularity`	For a historical SOW, the granularity of the history to store. In many cases, it is not necessary for AMPS to store all of the updates to the SOW. This parameter sets the resolution at which you can query history. For example, with a granularity of `1m`, AMPS will store the state of an updated messages no more frequently than once a minute.

Element	Description
	Used within the `History` element.

Even though the `RecordSize` defined may be smaller than the incoming message, the record will still be stored. Messages larger than the `RecordSize` will span multiple records. For example if the `RecordSize` is defined to be 128 bytes, and a message comes in that is 266 bytes in size, that record will be stored over 3 records. The maximum size for a single message is calculated as `RecordSize * IncrementSize`, or 1MB (whichever is larger). AMPS reports an error if a single message exceeds this size.

The listing in Example 4.1 is an example of using `TopicDefinition` to add a SOW topic to the AMPS configuration. One topic named `ORDERS` is defined as having key `/invoice`, `/customerId` and `MessageType` of `json`. The persistence file for this topic be saved in the `sow/ORDERS.json.sow` file. For every message published to the `ORDERS` topic, a unique key will be assigned to each record with a unique combination of the fields `invoice` and `customerId`. A second topic named `ALERTS` is also defined with a `MessageType` of `xml` keyed off of `/client/id`. The SOW persistence file for `ALERTS` is saved in the `sow/ALERTS.sow` file.

```
<SOW>
    <TopicDefinition>
        <FileName>sow/%n.sow</FileName>
        <Topic>ORDERS</Topic>
        <Key>/invoice</Key>
        <Key>/customerId</Key>
        <MessageType>json</MessageType>
        <RecordSize>512</RecordSize>
        <HashIndex>
           <Key>/region</Key>
        </HashIndex>
    </TopicDefinition>

    <TopicDefinition>
        <FileName>sow/%n.sow</FileName>
        <Topic>ALERTS</Topic>
        <Key>/alert/id</Key>
        <MessageType>xml</MessageType>
    </TopicDefinition>
</SOW>
```

Example 4.1. Sample SOW configuration

Topics are scoped by their respective message types and transports.

For example, two topics named Orders can be created one which supports `MessageType` of `json` and another which supports `MessageType` of `xml`.

Each of the `MessageType` entries that are defined for the `Orders` topic will require a unique `Transport` entry in the configuration file.

This means that messages published to the `Orders` topic must know the type of message they are sending (`fix` or `xml`) and the port defined by the transport.

Chapter 5. Advanced Topics

While there is much more content beyond the scope of this document, AMPS provides many of the following additional utilities and guides for you to learn about the many feature of AMPS.

5.1. Logging

AMPS supports logging to many different targets including the console, syslog, and files. Every error message within AMPS is uniquely identified and can be filtered out or explicitly included in the logger output. This chapter of the *AMPS User Guide* describes the AMPS logger configuration and the unique settings for each logging target.

5.2. Message Replay

AMPS supports a fully-queryable transaction log. You can configure the transaction log to keep a journal of incoming messages for one or more topics, and then replay those messages, in order, from any point in time. This capability is often used for historical analysis, as well as for clients to

The AMPS clients provide resumable subscription capability that works with the transaction log. Using this capability, you can create applications that ensure that clients never miss a message, even if the client is shut down and restarted.

5.3. Conflated Topics

To further reduce network bandwidth consumption, AMPS supports a form of SOW topic called a “conflated topic.” A conflated topic is a copy of one SOW topic into another with the ability to control the update interval. Changes to a message that occur between updates are conflated into a single message that represents the current state of the message.

To better see the value in a conflated topic, imagine a SOW topic called `ORDER_STATE` exists in an AMPS instance. `ORDER_STATE` messages are published frequently to the topic. Meanwhile, there are several subscribing clients that are watching updates to this topic and displaying the latest state in a GUI front-end.

If this GUI front-end only needs updates in five second intervals from the `ORDER_STATE` topic, then more frequent updates would be wasteful of network and client-side processing resources. To reduce network congestion, a conflating topic replica of the `ORDER_STATE` topic can be created which will contain a copy of `ORDER_STATE` updated in five second intervals. Only the changed records from `ORDER_STATE` will be copied to the conflating replica topic and then sent to the subscribing clients. Those records with multiple updates within the time interval will have their latest updated values sent during replication, resulting in substantial savings in bandwidth for single records with high update rates.

5.4. View Topics and Aggregation

AMPS contains a high-performance aggregation engine, which can be used to project one topic onto another, similar to the `CREATE VIEW` functionality found in most RDBMS software. Views can JOIN multiple topics together, including topics with different message types.

5.5. Historical SOW Query

AMPS allows you to configure a SOW topic to retain the historical state of the SOW, on a configurable granularity.You can then query for the state of the SOW at a point in time, and retrieve results from the saved state.

5.6. Utilities

AMPS provides several utilities that are not essential to message processing, but can be helpful in troubleshooting or tuning an AMPS instance:

- `amps_sow_dump` is used to inspect the contents of a SOW topic store.
- `amps_journal_dump` is used to examine the contents of an AMPS journal file during debugging and program tuning.
- `ampserr` is used to expand and examine error messages that may be observed in the logs. This utility allows a user to input a specific error code, or a class of error codes, examine the error message in more detail, and where applicable, view known solutions to similar issues.
- AMPS provides a command-line client, `spark`, as a useful tool for checking the status of the AMPS engine. The Spark client can also be used to run queries, place subscriptions, and publish data.

More information, including usage and examples, about each of these utilities can be found in the *AMPS Utilities User Guide*.

5.7. Monitoring Interface

AMPS provides a monitoring interface which contains information about the state of the host system (CPU, memory, disk and network) as well as statistics about the state of the AMPS instance it is monitoring (clients, SOW state, Journal State and more).

More information about the monitoring system provided in AMPS can be found in the *AMPS Monitoring Reference Guide*.

5.8. High Availability

The *High Availability* chapter in the *AMPS User Guide*will showcase the powerful High Availability features that AMPS provides. This chapter will first show how a Transaction Log can be configured to keep a journal of all messages published to a topic, then show how that journal can be used in different client recovery scenarios. Then we will show how Replication can be used to implement instance synchronization and failover strategies to guarantee that AMPS is always available, even in multiple locations.

Chapter 6. Next Steps

Now that you understand the basics of how AMPS works, you have two potential paths forward in your usage of the product:

- On one path, you may want to learn how to configure, deploy, and administer your own instance of AMPS. For this path, see the *User Guide*, which provides complete information for system administrators who are responsible for the deployment, availability and management of data to other users.
- Alternatively, you may need to develop an application to work with AMPS, using one of the Developer Guides for Java, Python, C++, or C#. For this path, download one of the evaluation kits from the AMPS developer page at http://www.crankuptheamps.com/developer.

The following sections provide more information about each of these paths and also briefly describes some use cases for AMPS.

6.1. Operation and Deployment

In preparing to deploy your instance of AMPS, you must size your host environment according to multiple dimensions: memory, storage, CPU, and network. The “Operation and Deployment” chapter in the *AMPS User Guide* provides guidelines and best practices for configuring the host environment. The chapter also specifies recommended settings for running AMPS on a Linux operating system.

6.2. Application Development

Each language-specific *Development Guide* explains how to install, configure, and develop applications that use AMPS. In order to develop applications using an AMPS client, you must understand the basic concepts of AMPS, such as *topics*, *subscriptions*, *messages* and *SOW*.

You will also need an installed and running AMPS server to use the product. Although you can type and compile programs that use AMPS without a running server, you will get the most benefit by running the programs against a working server. An evaluation version of AMPS is available from http://www.crankuptheamps.com/evaluate.

Index

V