

User Guide

60East Technologies™

Advanced Message Processing System™

3.3

Advanced Message Processing System™
60East Technologies™

Copyright © 2013 60East Technologies

All rights reserved. 60East, AMPS, and Advanced Message Processing System are trademarks of 60East Technologies, Inc. All other trademarks are the property of their respective owners.

PUBLICATION DATE: MARCH 4, 2013 [L^AT_EX 2_ε]

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

Contents

1	Introduction to 60East Technologies AMPS	1
1.1	Welcome	1
1.2	Product Overview	1
1.3	Software Requirements	2
1.4	Organization of this Manual	2
1.5	Document Conventions	3
1.6	Technical Support	4
2	Getting Started	6
2.1	Installing AMPS	6
2.2	Starting AMPS	7
2.3	Running the Demonstration Applications	7
2.4	Admin View of the AMPS Server	8
2.5	Interacting with AMPS Using Spark	9
2.6	FIX Messages - A Quick FIX Primer	9
2.7	Next Steps	10
3	Publish and Subscribe	11
3.1	Topics	12
3.2	Content	13
4	State of the World (SOW)	15
4.1	How Does the State of the World Work?	15
4.2	Queries	16
4.3	Configuration	16
5	SOW Queries	20
5.1	Simple SOW Queries	20
5.2	SOW Query-and-Subscribe	21
5.3	SOW Query Response Batching	22
6	Content Filtering	25
6.1	Syntax	25
7	Regular Expressions	30

7.1	Examples	30
8	Transports	34
8.1	Configuration	34
8.2	TCP/IP Transport	35
9	Message Types	38
9.1	Configuration	38
9.2	FIX	39
9.3	SOAP	40
10	Logging	42
10.1	Configuration	42
10.2	Log Messages	43
10.3	Log Levels	43
10.4	Logging to a File	45
10.5	Logging to a Compressed File	47
10.6	Logging to the Console	48
10.7	Logging to Syslog	49
10.8	Error Categories	50
10.9	Error discovery with ampserr	52
11	Event Topics	53
11.1	Client Status	53
11.2	SOW Statistics	54
11.3	Persisting Event Topic Data	56
12	Message Acknowledgment	57
12.1	Subscription Acknowledgment Messages	57
13	Topic Replicas	60
13.1	Configuration	61
14	View Topics	63
14.1	Example	63
15	Message Expiration	67
15.1	Usage	67
15.2	Example Message Lifecycle	68
16	Out of Focus Message Processing (OOF)	70
16.1	Usage	70
16.2	Example	71
16.3	Another Example	75
17	Utilities	79
17.1	amps_sow_dump	79
17.2	amps_journal_dump	81
17.3	ampserr	82
17.4	spark	84

18 Operation and Deployment	86
18.1 Capacity Planning	86
18.2 Linux Operating System Configuration	90
18.3 Best Practices	92
19 Monitoring Interface	96
19.1 Configuration	96
19.2 Time Range Selection	97
19.3 Output Formatting	98
20 High Availability	101
20.1 Transaction Log	101
20.2 Replication	102
20.3 Bookmarks	104
20.4 Publishing for High Availability	104
20.5 Subscribing for High Availability	106
20.6 Deployment Examples	108
20.7 Potential Points of Failure	114
20.8 Configuration	115
21 Sample Use Cases	119
21.1 View Server Use Case	119
A Header Field Reference	125
A.1 FIX Message Header - Sorted by Value	125
A.2 FIX Message Header - Sorted by Name	126
A.3 XML Message Header - Sorted by Name	127
A.4 Header Fields - Sorted by Name	128
B Command Reference	132
B.1 delta_publish	132
B.2 delta_subscribe	135
B.3 logon	137
B.4 publish	139
B.5 sow_and_delta_subscribe	141
B.6 sow_and_subscribe	144
B.7 sow_delete	146
B.8 sow	149
B.9 start_timer	151
B.10 stop_timer	152
B.11 subscribe	153
B.12 unsubscribe	156
C Configuration Reference	158
C.1 AMPS Configuration Basics	158
C.2 Generating a Configuration File	163
C.3 Features	164
D Monitoring Interface Reference	178
D.1 Host Interface	178
D.2 Instance Interface	181

E Authentication and Entitlements	191
E.1 Configuration	191
E.2 AMPS Administration	192
E.3 AMPS Guarantees	193
E.4 Authentication Interface	194
E.5 Entitlements Module interface	196
E.6 Additional Notes	199
F Glossary	200

Chapter 1

Introduction to 60East Technologies AMPS

1.1 Welcome

Thank you for choosing the Advanced Message Processing System (AMPS) from 60East Technologies. AMPS is a feature-rich publish and subscribe message processing system that delivers previously unattainable low-latency and high-throughput performance to users.

1.2 Product Overview

AMPS is a modern publish and subscribe engine designed specifically for next generation computing environments. It is intended to allow the realization of scalable high-throughput, low-latency messaging required in real-time deployments such as in financial services. 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.

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 its consumption by end-user clients.

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

- Topic and content based publish and subscribe
- Native FIX and XML message support

- 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
- Delta publish and subscriptions
- Basic CEP capabilities for real-time computation and analysis
- Replication for High-Availability
- Message replay functionality

1.3 Software Requirements

AMPS is supported under the following platforms:

- Linux 64-bit (2.6 kernel or later) on x86 compatible processors ¹

1.4 Organization of this Manual

This manual is divided into the following chapters:

- [Chapter 1](#) --- Introduction to AMPS; (this chapter) describes the product, provides information on using this manual efficiently, and explains how to obtain technical support.
- [Chapter 2](#) --- AMPS Basics; covers installation, basic configuration, operation and usage. Start here if you want to start using AMPS immediately.
- [Chapter 3](#) through [Chapter 21](#) contain feature-specific information:
 - ◇ [Chapter 3](#) --- Publishing and Subscribing
 - ◇ [Chapter 4](#) --- State of the World (SOW)
 - ◇ [Chapter 5](#) --- SOW Queries
 - ◇ [Chapter 6](#) --- Content Filtering
 - ◇ [Chapter 7](#) --- Regular Expressions
 - ◇ [Chapter 8](#) --- Transports
 - ◇ [Chapter 9](#) --- Message Types
 - ◇ [Chapter 10](#) --- Logging
 - ◇ [Chapter 11](#) --- AMPS Event Topics



¹While 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.

- ◇ Chapter 12 --- Message Acknowledgment
- ◇ Chapter 13 --- Topic Replicas
- ◇ Chapter 14 --- View Topics
- ◇ Chapter 15 --- Message Expiration
- ◇ Chapter 16 --- OOF - Out of Focus Messages
- ◇ Chapter 17 --- Utilities
- ◇ Chapter 18 --- Operation and Deployment
- ◇ Chapter 19 --- Monitoring Interface
- ◇ Chapter 20 --- High Availability
- ◇ Chapter 21 --- Sample Use Cases
- Appendix A --- Header Field Reference
- Appendix B --- AMPS Command Reference
- Appendix C --- AMPS Configuration Reference
- Appendix D --- Monitoring Interface Reference

1.5 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 fragments
<i>variable</i>	variables within commands or configuration
	usage tip or extra information
	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. A content filter that is long will show arrows to indicate word wrapping as in this example:

```
(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
```

XML fragments will be formatted with line numbers as in the following example:

```
1 <XML>
2   <AMPS>fast</AMPS>
3 </XML>
```

1.6 Technical 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.

1.6.1 Obtaining Support

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.
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 support@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 . TIMESTAMP . TAG
```

Each AMPS version number component has the following breakdown

Table 1.2: Version Number Components

Component	Description
MAJOR	Ticks when there are changes in functionality, file formats, configs, or deprecated functionality.
MINOR	Ticks when new functionality is added.
MAINTENANCE	Ticks with standard bug fixing, maintenance, small features and enhancements.
TIMESTAMP	Proprietary build timestamp.
TAG	Identifier that corresponds to precise code used in the release.

The packaging of a release will also take into account the source control branch, so that if anything other than `master` is used the branch name will show up in the `cpack` distribution.

1.6.2 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
Support Phone:	(646) 392-8267

Chapter 2

Getting Started

Chapter 2 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	Client APIs, <code>spark</code> sample client and examples
bin	AMPS engine binaries and utilities
demos	Demo applications
docs	Documentation
lib	Library dependencies

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.

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 3.0.0 - Copyright (c) 2006 - 2011 60East ↵
    Technologies, Inc.
    (Built: Oct 16 2011 13:53:41)

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

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

2.3 Running the Demonstration Applications

After starting the AMPS instance, AMPS is now ready to accept published data and accept subscriptions. In the `$AMPSDIR/demos` directory there are a couple of demonstration programs that publish and subscribe to AMPS. Both the XML and FIX demo programs establish a subscriber and a publisher. The `$AMPSDIR/demos` directory includes the following files:

File	Description
<code>amps_fix_demo</code>	A FIX publisher / subscriber application
<code>amps_xml_demo</code>	An XML publisher / subscriber application
<code>client_monitor</code>	An application that listens for and prints client events and activity

The `amps_fix_demo` and `amps_xml_demo` applications contain a publisher and a subscriber. The publisher sends a `start_timer` command, then it publishes 100,000 messages and finishes by sending a `stop_timer` command. When

the `stop_timer` response is received from AMPS, the publisher will repeat publishing the data until you stop the program. On each of the publisher's iterations, the amps engine will print out a status message with the performance statistics.

The subscriber will subscribe with a topic and content filter that looks like:

```
(/FIXXML/Order/Instrmt/@Sym = 'IBM') AND
 (/FIXXML/Order/OrdQty/@Qty > 9999)
```

The FIX equivalent is:

```
(/20 = 'IBM') AND (/21 > 9999)
```



The examples contain fictional data only to illustrate how filtering works.

In the demo configuration, there are two ports opened and listening for client connections: FIX clients on port 9004 and XML clients on 9005. Therefore, to start the FIX demo application run:

```
$AMPSDIR/demos/amps_fix_demo -s host:9004
```

Or, for XML run:

```
$AMPSDIR/demos/amps_xml_demo -s host:9005
```

Where *host* is the IP address of the host running AMPS. If the AMPS engine instance is running on the same host, use "localhost" for the *host*.

After starting the demo applications, the console in which AMPS was started will have performance metrics and statistics printed to the screen. Additionally, the metrics from the administration and statistics interface can be observed on port 9090 of the host running the AMPS engine instance.

To shut down the AMPS engine instance: If the engine is running in the foreground of the console, the issuing a `Ctrl+C` will cause AMPS to shutdown safely. If the AMPS engine instance is running in the background, using the `kill -SIGTERM` followed by the `pid` of the AMPS engine instance will safely shut it down.

2.4 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. (For more information on

the monitoring capabilities, please see [Chapter 19](#).) For the demo configuration, try connecting to `http://localhost:9090`.

2.5 Interacting with AMPS Using Spark

AMPS provides the 'spark' utility as a command line interface to interacting with an AMPS server. This lets you execute commands like 'subscribe', 'publish', 'sow', 'sow_and_subscribe' and 'sow_delete'.

The spark utility is implemented for Java, C++, C# and Python, which are the languages supported by the AMPS Client. There is a reference spark client and source code for Java, C++, C# and Python.

The command-line syntax is identical between all versions of spark. For these examples, we will be using the Java implementation of spark. We will also be using FIX for the message type in our examples.

Go to the `$AMPS_INSTALL/api/client/java` directory and run the following command:

```
./spark publish -server localhost:9004 -type fix -topic↔  
sampleTopic
```

The spark program starts and waits for input from the command line. Type the message body you'd like to test. For example:

```
1=a^A2=b^A3=c^A
```

If no errors are returned, we'll assume the FIX message will be published to the 'sampleTopic' topic on the server.

The spark utility also supports subscribing, which allows you to listen to messages on a specified topic. If you would like to only receive messages from the 'sampleTopic' topic we used in the publish example, you can do that with the following spark command:

```
./spark subscribe -server localhost:9004 -type fix -↔  
topic sampleTopic
```

This subscribe command will write all messages sent to this topic to stdout from the spark command.

For more information about the spark utility, see [Chapter 17](#).

2.6 FIX Messages - A Quick FIX Primer

AMPS includes support for XML, FIX and NVFIX messages. This section is going to focus on FIX as the primary message type. The FIX message type is a simple message format that uses key-value pairs to construct message

content and uses special characters to separate message content. In a FIX message, all keys are numeric and values are alpha-numeric. Separators are usually obscure or infrequently used characters, such as the ASCII 01 character to denote the header separator. In this section, the following characters will be used to denote special separators:

ASCII Char	Control Key	FIX Use
01	^A	Field Separator
02	^B	Header Separator
03	^C	Message Separator

Using what we have described so far, a simple FIX message could look like:

```
20000=publish^A20005=sampleTopic^A^B1=a^A2=d^A4=e^A
```

This creates a message that has a 'command' (FIX field 20000) of 'publish' to 'topic' (FIX field 20005) with the value of 'sampleTopic' as the message header. This message has three fields as part of its body: field 1 is set to 'a', field 2 is set to 'd' and field 4 is set to 'e'.

For more information about the message types used in AMPS, see [Chapter 9](#).

2.7 Next Steps

The next step is to configure your own instance of AMPS to meet your messaging needs. The AMPS configuration is covered in more detail in [Appendix C](#).

After you have successfully configured your own instance, there are two paths where you can go next.

One path is to continue using this guide and learn how to configure, administer and customize AMPS in depth so that it may meet the needs of your deployment. If you are a system administrator who is responsible for the deployment, availability and management of data to other users, then you may want to focus on this User Guide.

The other path introduces the AMPS Client API for Java or Python. This path is targeted at software developers looking to integrate AMPS into their own solutions. The Java or Python AMPS Developer Guides are also available within the AMPS distribution in the `docs` directory.

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.

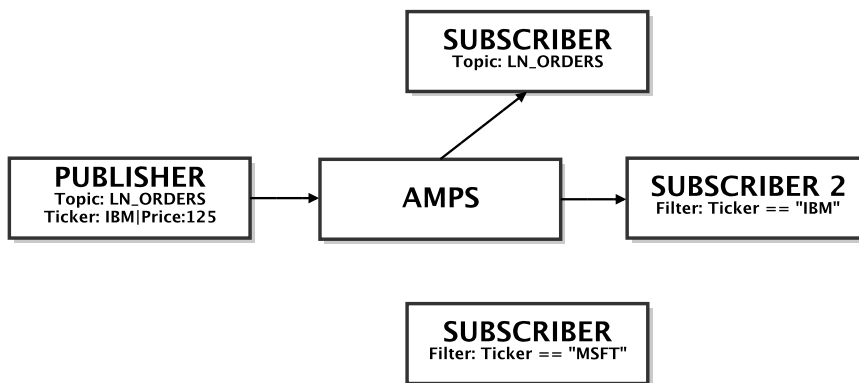


Figure 3.1: Publish and Subscribe

AMPS can route messages from publishers to subscribers using a topic identifier and/or content within the message's payload. For example, in [Chapter 3](#), 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 they are subscribed to.

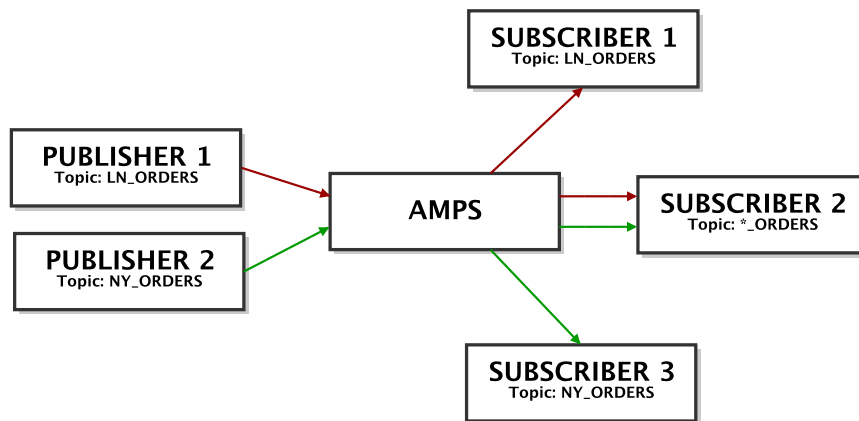


Figure 3.2: Topic Based Pub/Sub

For example, in [Section 3.1](#), there are two publishers: Publisher 1 and Publisher 2 which publish to 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.

3.1.1 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 [Chapter 7](#).

3.2 Content

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

The ability to use content filters greatly reduces the problem of over subscription 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.

Content-based Pub/Sub is somewhat analogous to database queries with a `WHERE` clause specified. Topics can be considered tables into which rows are inserted. 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 Pub/Sub is analogous to a `SELECT` on a table with no limiting `WHERE` clause.

Example XML Filter:

```
(/FIXML/Order/Instrmt/@Sym = 'IBM') AND (/FIXML/Order/↵
  @Px >= 90.00 AND /FIXML/Order/@Px < 91.0)
```

Example FIX Filter:

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

For more on how content is handled within AMPS, check out the chapter on specific message types in [Chapter 6](#).



It is recommended that a relatively small set of topics be used to categorize messages at a high level and content filters used to retrieve specific data published to those topics. Examples of good, broad topic choices:

trades, positions, MarketData, Europe, alerts

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, this is accomplished through the definition of key fields specified in the AMPS configuration which uniquely identify messages within the SOW.

Key definitions are similar to primary keys in a relational database. In AMPS, these unique keys are defined as one or more XPath. The values of these XPath for a given message uniquely identify the message. Subsequent messages with the same values for the key XPath will be considered the same and will overwrite the previously stored message. Providing a key to define message uniqueness enables a SOW queries for a topic.

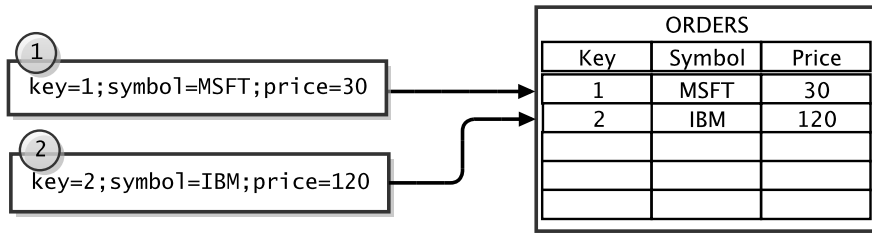


Figure 4.1: A SOW topic named `ORDERS` with a key definition of `/Key`

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.

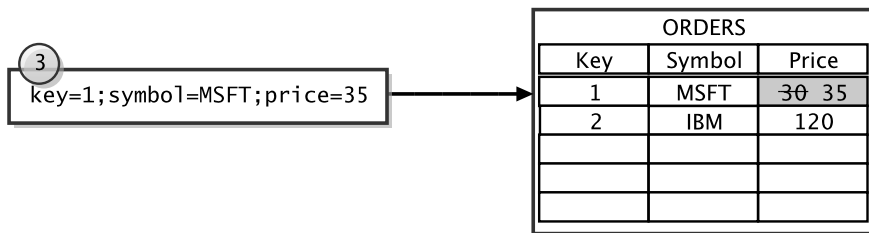


Figure 4.2: Updating the `MSFT` record by matching incoming message keys

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 straight topic or a regular expression pattern. For more information on issuing queries, please see [Chapter 5](#).

4.3 Configuration

Topics where SOW persistence is desired can be individually configured within the `TopicMetaData` section of the configuration file. Each topic will be defined with a `TopicDefinition` section enclosed within `TopicMetaData`. [Table 4.1](#) contains a description of the attributes that can be configured per topic.

Table 4.1: Topic Definition Configuration Description

Attribute	Description
FileName	The path name prefix of where to store the SOW topic files. The FileName attribute supports several name specifiers which can be read about in AMPS Configuration File Special Characters . Example: <code>%n.sow</code> will write sow messages for topic "orders" to <code>orders.sow</code>
Key	An XPath String that is used to define the uniqueness of a message published to the topic. A single topic must have at least one Key defined. When more than one Key attribute is defined, then all Key attributes are used to determine a record's uniqueness.
MessageType	The message type for this topic. Valid message types: <code>xml</code> , <code>fix</code>).
Topic	Specifies the topic to store into the SOW.
Duration	Possible values are <code>persistent</code> or <code>transient</code> . Duration defines how a SOW topic store behaves when an AMPS instance is restarted. If Duration is <code>persistent</code> then existing SOW records will be recreated each time AMPS is restarted, while <code>transient</code> specifies that existing SOW records will be deleted. Default is <code>persistent</code>
Expiration	The interval during which a message persists in a SOW cache before it is deleted. The minimum interval is 1 second, and by default topics do not have expiration enabled. See Table C.1 for details.
IncrementSize	The number of records to add when the SOW cache needs to grow. The valid range is between 128 and 10,000,000 with a default of 10,000.
InitialSize	The initial number of records in the SOW cache for this topic. The valid range is between 128 and 10,000,000 with a default of 10,000.
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.
RecordSize	The record size of the SOW cache for this topic. The minimum is 128B while the maximum is 16KB, with a default of 512. See Table C.2 for configuration details.



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 listing in [Table 4.3](#) is an example of using `TopicDefinition` to add a SOW topic to the AMPS configuration. One topic named `ORDERS` is defined as having key `/55, /109` and `MessageType` of `fix`. The persistence file for this topic be saved in the `sow/ORDERS.sow` file. For every message published to the `ORDERS` topic, a unique key will be assigned to each record with a unique combination of tags 55 and 109. 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.

```

1 <TopicMetaData>
2   <TopicDefinition>
3     <FileName>sow/%n.sow</FileName>
4     <Topic>ORDERS</Topic>
5     <Key>/55</Key>
6     <Key>/109</Key>
7     <MessageType>fix</MessageType>
8     <RecordSize>512</RecordSize>
9   </TopicDefinition>
10
11  <TopicDefinition>
12    <FileName>sow/%n.sow</FileName>
13    <Topic>ALERTS</Topic>
14    <Key>/alert/id</Key>
15    <MessageType>xml</MessageType>
16  </TopicDefinition>
17 </TopicMetaData>

```



Topics are scoped by their respective message types and transports.

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

Each of the `MessageType` entries that are defined for the `litOrders` 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

SOW Queries

When SOW topics are configured inside an AMPS instance, clients can issue SOW queries to AMPS to retrieve all of the messages matching 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 straight topic or a regular expression pattern.

5.1 Simple SOW Queries

A client can issue a query by sending AMPS a `sow` command and specifying an AMPS topic. Optionally a filter can be used to further refine the query results. When AMPS receives the `sow` command request, it will validate the filter and start executing the query. When returning a query result back to the client, AMPS will package the `sow` results into a `sow` record group by first sending a `group_begin` message followed by the matching SOW records, if any, and finally indicating that all records have been sent by terminating with a `group_end` message. The message flow is provided as a sequence diagram in [Figure 5.1](#)

For purposes of correlating a query request to its result, each query command can specify a `QueryId`. The `QueryId` specified will be returned as part of the response that is delivered back to the client. The `group_begin` and `group_end` messages will have the `QueryId` attribute set to the value provided by the client. The client specified `QueryId` is what the client can use to correlate query commands and responses coming from the AMPS engine.



The ordering of records returned by a SOW query is undefined.

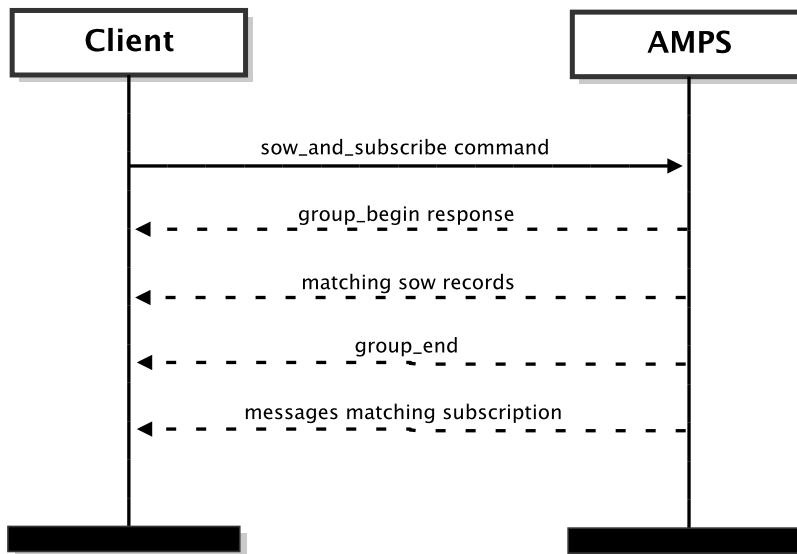


Figure 5.1: SOW Query Sequence Diagram

5.2 SOW Query-and-Subscribe

AMPS has a special command that will execute a query and place a subscription at the same time to prevent a gap between the query and subscription where messages can be lost. Without a command like this, it is difficult to reproduce the SOW state locally on a client.

For an example, this command is useful for recreating part of the SOW in a local cache and keeping it up to date. Without a special command to place the query and subscription at the same moment, a client is left with two options:

1. issue the query request, process the query results, and then place the subscription, which misses any records published between the time when the query and subscription were placed; or
2. place the subscription and then issue the query request, which could send messages placed between the subscription and query twice. Instead of coping with those drawbacks, the AMPS `sow_and_subscribe` command allows clients to place a query and get the streaming updates to matching messages in a single command.

In a `sow_and_subscribe` command, AMPS behaves as if the SOW command and subscription are placed at the exact same moment. The SOW

query will be sent *before* any messages from the subscription are sent to the client. Additionally, any new publishes that come into AMPS that match the `sow_and_subscribe` filtering criteria and come in after the query started will be sent after the query finishes (and the query will not include those messages.)

The message flow as a sequence diagram for `sow_and_subscribe` commands is contained in [Figure 5.2](#)

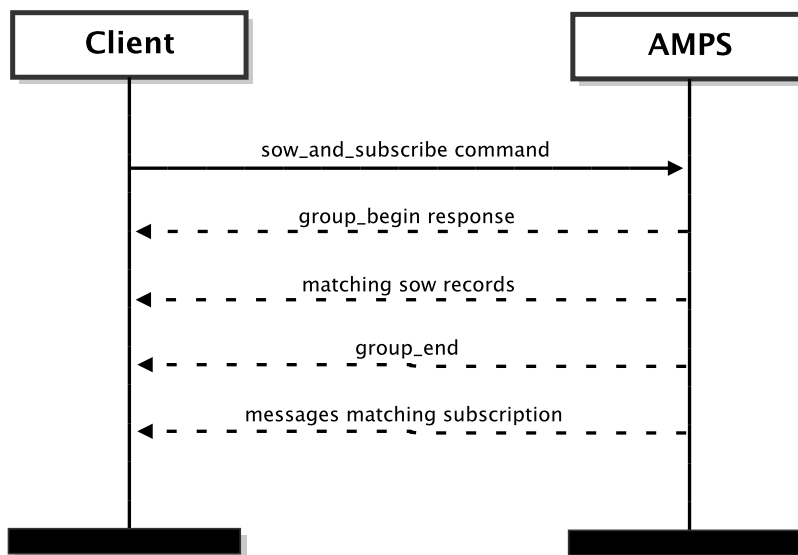


Figure 5.2: SOW-And-Subscribe Query Sequence Diagram

5.3 SOW Query Response Batching

The number of messages returned between the `group_begin` and `group_end` messages depends on the number of matching messages and the value of the `BatchSize` parameter that can be specified in the query command. The default `BatchSize` value is 1, meaning 1 record is returned per containing `sow` message in the response. While the SOW is scanned for matches it will attempt to send them to the client in batches according to the value of the `BatchSize` value. The `BatchSize` is the maximum number that will be returned within a single message payload. Each message returned for a given query command will contain a `BatchSize` value in its header to indicate the number of messages in the batch. This number will be anywhere from 1 to `BatchSize`.



When issuing a `sow_and_subscribe` command AMPS will return a `group_begin` and `group_end` segment of messages before beginning the live subscription sequence of the query. This is also true when an `sow_and_subscribe` command is issued against a non-SOW topic. In this later case, the `group_begin` and `group_end` will contain no messages.

Using a `BatchSize` greater than 1 can yield greater performance when querying a large number of small records. For example, over an Ethernet connection where the packet size may only be 1500 bytes, consuming 1 million 120 byte messages could take 1 million packets and the efficiency of the network would only be around 10%. In this same scenario, we could use a `BatchSize` of 10 to pack 10 messages into a single message of approximately 1230 (30 extra bytes for the `sow` message header) bytes and boost the network efficiency to 82%.



Using an appropriate `BatchSize` parameter is critical to achieve the maximum query performance with a large number of small messages.

Each message within the batch will contain id and key values to help identify each message that is returned as part of the overall response.

For XML, the format of the response is:

```

1 <?xml version="1.0" encoding="iso-8859-1"?>
2 <SOAP-ENV:Envelope>
3   <SOAP-ENV:Header>
4     <Cmd>sow</Cmd>
5     <TxmTm>20080210-17:16:46.066-0500</TxmTm>
6     <QId>100</QId>
7     <GrpSqNum>1</GrpSqNum>
8     <BtchSz>5</BtchSz>
9     <Tpc>order</Tpc>
10  </SOAP-ENV:Header>
11  <SOAP-ENV:Body>
12    <Msg key="143101935596417" len="120"> ... </Msg>
13    <Msg key="86456484160208" len="125"> ... </Msg>
14    <Msg key="18307726844082" len="128"> ... </Msg>
15    <Msg key="15874572074104" len="118"> ... </Msg>
16    <Msg key="61711462299630" len="166"> ... </Msg>
17  </SOAP-ENV:Body>
18 </SOAP-ENV:Envelope>

```

For FIX, the format has the following form:

```
{sow header}①{message header}①{message data}②{↵
message header}①{message data}② ...
```

where:

- ① header separator
- ② message separator

Each message header will contain the `SowKey` and the `MessageLength` attributes. The `MessageLength` is intended to help clients parse the response with minimal effort. It indicates the length of the message data contained in the message.

The following is an example FIX message SOW query response message:

```
20000=sow①20004=20080210-17:16:46.066-0500①20007=fix↵
①20019=100①\newline20060=1①20023=5①20005=order↵
①②20059=1①20058=128①②fix message data③
```

where:

- ① header separator
- ② field separator
- ③ message separator



Care should be taken when issuing queries that return large results. When contemplating the usage of large queries and how that impacts system reliability and performance, please look to the [Slow Clients](#) section for more information.

For more information on executing queries, please look at the `sow` and `sow_and_subscribe` commands in the [Command Reference](#) chapter.

Chapter 6

Content Filtering

AMPS allows a subscriber to specify a content filter using syntax similar to that of SQL-92's WHERE clause. Content filters are used to provide a greater level of selectivity in messages sent over a subscription than provided by topic alone. When using a content filter, only the messages matching the requested topic and the content filter are delivered to the subscribing client.

6.1 Syntax

AMPS implements content filtering using expressions that combine SQL-92 and XPath. Instead of table columns, XPaths are used to identify content fields within a message. The syntax of the filter expression is based on a subset of the SQL-92 search condition syntax.

A content filter is made up of one or more Boolean predicates, joined together by logical operators and grouped using parentheses. For example:

```
(expression1 OR expression2 AND expression3) OR (↔  
expression4 AND NOT expression5) ...
```

A content filter is evaluated left to right in precedence order. So in this example, expression2 will be evaluated followed by expression3 (since AND has higher precedence than OR), and if they evaluate to false, then expression1 will be evaluated and so on.

Expressions are made up of literal values, identifiers (XPath that refer to message content values), conditional operators, and arithmetic operators.

Literals

String literals are indicated with single or double quotes. For example:

```
/FIXML/Order/Instrmt/@Sym = 'IBM'
```

AMPS supports the following escape sequences within string literals:

Table 6.1: Escape Sequences

Escape Sequence	Definition
<code>\a</code>	alert.
<code>\b</code>	Backspace.
<code>\t</code>	Horizontal tab.
<code>\n</code>	Newline.
<code>\f</code>	Form Feed.
<code>\r</code>	Carriage Return.
<code>\xHH</code>	Hexadecimal digit where H is (0..9,a..f,A..F).
<code>\OOO</code>	Octal Digit (0..7)

Additionally, any character which follows a backslash will be treated as a literal character.

Numeric literals are either integer values or floating-point values. Integer values are all numerals, with no decimal point, and can have a value in the same range as a 64-bit integer. For example:

```
42
149
-273
```

Floating-point literals are all numerals with a decimal point:

```
3.1415926535
98.6
-273.0
```

or, in scientific notation:

```
31.4e-1
6.022E23
2.998e8
```

Literals can also be the Boolean values `true` or `false`.

Logical Operators

The logical operators are `NOT`, `AND`, and `OR`, in order of precedence. These operators have the usual Boolean logic semantics.


```
/FIXML/Order/Instrmt/@Sym = 'IBM' OR /FIXML/Order/↔
Instrmt/@Sym = 'MSFT'
```

Arithmetic Operators

AMPS supports the arithmetic operators +, -, *, /, %, and MOD in expressions. The result of arithmetic operators where one of the operands is NULL is undefined and evaluates to NULL. Examples of filter expressions using arithmetic operators:

```
/6 * /14 < 1000
/Order/@Qty * /Order/@Prc >= 1000000
```



When using division, be careful about the placement of the / operator. Since this operator is used in the XPath expression as well as for division, it is important to separate the division operator with whitespace to prevent interpretation as a XPath expression.

Comparison Operators

The comparison operators can be loosely grouped into equality comparisons and range comparisons. The basic equality comparison operators, in precedence order, are ==, =, >, >=, <, <=, !=, and <>. If these binary operators are applied to two operands of different types, then AMPS will try to perform a conversion that permits comparison.

There are also set and range comparison operators. The BETWEEN operator can be used to check the range values.



The range used in the BETWEEN operator is inclusive of both operands meaning the expression /A BETWEEN 0 AND 100 is equivalent to /A >= 0 AND /A <= 100

For example:

```
/FIXML/Order/OrdQty/@Qty BETWEEN 0 AND 10000
/FIXML/Order/@Px NOT BETWEEN 90.0 AND 90.5
```

The IN operator can be used to perform membership operations on sets of values:

```
/Trade/OwnerID NOT IN ('JMB', 'BLH', 'CJB')
/21964 IN (/14*5, /6*/14, 1000, 2000)
```

There is also a string comparison operator, LIKE, that allows for regular expression matching on string values. A pattern is used for the right side of the LIKE operator. For more on regular expressions and the LIKE comparison operator, please see the [Regular Expressions](#) chapter.

Conditional Operators

AMPS contains support for a ternary conditional IF operator which allows for a Boolean condition to be evaluated to true or false, and will return one of the two parameters. The general format of the IF statement is

```
IF ( BOOLEAN_CONDITIONAL, VALUE_TRUE, VALUE_FALSE)
```

In this example, the BOOLEAN_CONDITIONAL will be evaluated, and if the result is true, the VALUE_TRUE value will be returned otherwise the VALUE_FALSE will be returned.

For example:

```
SUM( IF( ( (/FIXML/Order/OrdQty/@Qty > 500) AND
(/FIXML/Order/Instrmt/@Sym = 'MSFT') ), 1, 0 ))
```

In the above example, we are looking for the total number of orders that have been placed where the symbol is MSFT and the order contains a quantity more than 500.

The IF can also be used to evaluate results to determine if results are NULL or NaN.

For example:

```
SUM( /FIXML/Order/Instrmt/@Qty * IF( /FIXML/Order/↵
Instrmt/@Price IS NOT NULL, 1, 0) )
```

NULL, NaN and IS NULL

XPath expressions that evaluate to a field references that is empty or non-existent are considered to be NULL. In numeric expressions where the operands or results are not a valid number evaluate to NaN (not a number). The rules for applying the AND and OR operators against NULL and NaN values are outlined in [Table 6.2](#) and [Table 6.3](#).

Operand 1	Operand 2	Result
TRUE	NULL	NULL
FALSE	NULL	FALSE
NULL	NULL	NULL

Table 6.2: Logical AND with NULL/NaN Values

Operand 1	Operand 2	Result
TRUE	NULL	TRUE
FALSE	NULL	NULL
NULL	NULL	NULL

Table 6.3: Logical OR with NULL/NaN Values.

The `NOT` operator applied to a `NULL` value is `NULL`, or "Unknown" as well. The only way to check for the existence of a `NULL` value reliably is to use the `IS NULL` predicate. There exists a `IS NAN` predicate as well for checking that a value is NaN (not a number.)



To reliably check for existence of a `NULL` value, you must use the `IS NULL` predicate such as the filter: `/Order/Comment IS NULL`

Chapter 7

Regular Expressions

AMPS supports regular expression matching on topics and within content filters. Regular expressions are implemented in AMPS using the Perl-Compatible Regular Expressions (PCRE) library. For a complete definition of the supported regular expression syntax, please refer to:

<http://perldoc.perl.org/perlre.html>.

7.1 Examples

Here's an example of a content filter for messages that will match any message meeting the following criteria:

- Symbols of 2 or 3 characters starting with "IB"
- Prices starting with "90"
- Prices less than 91.

and, the corresponding content filter:

```
1 (/FIXML/Order/Instrmt/@Sym LIKE "^IB.?$") AND (/FIXML/↵
   Order/@Px LIKE "^90\..*"
2 AND /FIXML/Order/@Px < 91.0)
```

Listing 7.1: Filter Regular Expression Example

The following tables (Table 7.1, Table 7.2, and Table 7.3) contain a brief summary of special characters and constructs available within regular expressions.

Here are more examples of using regular expressions within AMPS.

Use `(?i)` to enable case-insensitive searching. For example, the following filter will be true regardless if `/client/country` contains "US" or "us".

```
(/client/country LIKE "(?i)^us$")
```

Listing 7.2: Case Insensitive Regular Expression

To match messages where tag 55 has a TRADE suffix, use the following filter:

```
(/55 LIKE "TRADE$")
```

Listing 7.3: Suffix Matching Regular Expression

To match messages where tag 109 has a US prefix, but a TRADE suffix with case insensitive comparisons, use the following filter:

```
(/109 LIKE "(?i)^US.*TRADE$")
```

Listing 7.4: Case Insensitive Prefix Regular Expression

Table 7.1: Regular Expression Meta-characters

Characters	Meaning
^	Beginning of string
\$	End of string
.	Any character except a newline
*	Match previous 0 or more times
+	Match previous 1 or more times
?	Match previous 0 or 1 times
	The previous is an alternative to the following
()	Grouping of expression
[]	Set of characters
{ }	Repetition modifier
\	Escape for special characters

Table 7.2: Regular Expression Repetition Constructs

Construct	Meaning
a^*	Zero or more a 's
a^+	One or more a 's
$a^?$	Zero or one a 's
$a\{m\}$	Exactly m a 's
$a\{m,\}$	At least m a 's
$a\{m,n\}$	At least m , but no more than n a 's

Table 7.3: Regular Expression Behavior Modifiers

Modifier	Meaning
i	Case insensitive search
m	Multi-line search
s	Any character (including newlines) can be matched by a . character
x	Unescaped white space is ignored in the pattern.
A	Constrain the pattern to only match the beginning of a string.
U	Make the quantifiers non-greedy by default (the quantifiers are greedy and try to match as much as possible by default.)

7.1.1 Raw Strings

AMPS additionally provides support for raw strings which are strings prefixed by an 'r' or 'R' character. Raw strings use different rules for how a backslash escape sequence is interpreted by the parser.

In the example below, the raw string - noted by the `r` character in the second operand of the `LIKE` predicate (Listing 7.5) - will cause the results to parse the same as example which does not implement the raw string in the `LIKE` predicate (Listing 7.6). In this example we are querying for string that contains the programming language named `C++`. In the regular string, we are required to escape the '+' character since it is also use by AMPS as the "match previous 1 or more times" regular expression character. In the raw string we can use `r'C++'` to search for the string and not have to escape the special '+' character.

```
1 /FIXML/Language LIKE r'C++'
```

Listing 7.5: Raw String Example

```
1 /FIXML/Language LIKE 'C\+\+'
```

Listing 7.6: Regular String Example

7.1.2 Topic Queries

As mentioned previously, AMPS supports regular expression filtering for SOW topics, in addition to content filters. Topic filtering follows similarly to the regular expressions demonstrated in content filtering with a `LIKE` clause.

Topic filtering will search for all matching records stored in an AMPS SOW where the name of the SOW matches the regular expression used in the Topic name of the query. For example, if your AMPS configuration has three SOW topics, `Topic_A`, `Topic_B` and `Topic_C` and you wish to search for all messages in all

of your SOW topics for records where the `Name` field is equal to "Bob", then you could use the following `FIX sow` command to perform this:

```
1 /20000=sow^A/20005="Topic_.*"^A^B/20006="/FIXML/@Name=<↵
   'Bob' "^A
```

Listing 7.7: Topic Regular Expression



Results returned when performing a topic regular expression query will follow "configuration order" - meaning that the topics will be searched in the order that they appear in your AMPS configuration file. Using the above query example [Listing 7.7](#) with `Topic_A`, `Topic_B` and `Topic_C`, if the configuration file has these topics in that exact order, the results will be returned first from `Topic_A`, then from `Topic_B` and finally the results from `Topic_C`. As with other queries, AMPS does not make any guarantees about the ordering of results within any given topic query.

Chapter 8

Transports

AMPS is transport agnostic by design, which allows the support of any message delivery protocol. The current release AMPS contains support for TCP/IP transports.

Transports are independent of the message type. For example, AMPS could consume messages published via one TCP transport and deliver messages to downstream subscribers via another TCP/IP transport, as long as they are using the same message type.

8.1 Configuration

AMPS Transports are configured through the Transport section of the configuration file. For example, [Listing 8.1](#) is an example Transport section for using the TCP/IP transport with the `fix` message type.

```
1 <Transports>
2   <Transport>
3     <Name>fix-tcp</Name>
4     <Type>tcp</Type>
5     <MessageType>fix</MessageType>
6     <InetAddr>9004</InetAddr>
7     <ReuseAddr>true</ReuseAddr>
8   </Transport>
9 </Transports>
```

Listing 8.1: TCP with FIX transport example.

The following sections describe the transports shipped with AMPS.

8.2 TCP/IP Transport

The `tcp` transport is configured by adding a `Transport` section to the configuration and specifying `tcp` as the `Type`. Listing [Listing 8.2](#) contains an example of a `tcp` transport.

```

1 <Transports>
2   <Transport>
3     <Name>fix-tcp</Name>
4     <Type>tcp</Type>
5     <MessageType>fix</MessageType>
6     <InetAddr>9004</InetAddr>
7     <ReuseAddr>true</ReuseAddr>
8   </Transport>
9 </Transports>

```

Listing 8.2: TCP/IP configuration transport example

The parameters allowed by the `tcp` transport are in [Table 8.1](#).

Table 8.1: TCP/IP Transport configuration parameters

Parameter	Description
Name	Name of transport, can be anything, but useful for error log messages and admin information, especially when there are multiple transports of the same type.
Type	Specifies the transport type ('tcp' in this case).
MessageType	Message type used by the transport.
InetAddr	Address to bind to or if an integer, the port to bind to.
ReuseAddr	<code>true</code> or <code>false</code> , defaults to <code>false</code> and determines if AMPS should be allowed to bind to an address that is in the <code>TIME_WAIT</code> state.
ClientOffline	<code>enabled</code> or <code>disabled</code> , defaults to <code>disabled</code> . When enabled "slow" clients that breach the <code>ClientBufferThreshold</code> in data awaiting consumption will be offlined to a file. Please see Section 8.2.1 for more details.
ClientOfflineDirectory	The directory to store the offlining data to. Ideally, this should be on a fast storage device. The default directory is <code>/var/tmp</code> .
ClientBufferThreshold	The threshold in number of bytes to start offlining a "slow" client.
ClientOfflineThreshold	The number of messages to offline before AMPS will consider disconnecting a "slow" client.

Continued on next page

Table 8.1 -- continued from previous page

Parameter	Description
SlowClientDisconnect	<code>true</code> or <code>false</code> , defaults to <code>false</code> and determines if slow clients can be disconnected when they fall behind by the number of messages specified in <code>ClientOfflineThreshold</code> .

8.2.1 Slow Client Offlining

AMPS sends data as quickly as possible to downstream subscribing clients, in such scenarios it can become overwhelming for a client to keep up when it has oversubscribed or is unable to consume messages at the rate AMPS sends them. This mismatch is best seen when a client executes a SOW query that returns a large result set. During a SOW query, AMPS is capable of collecting and queueing the result set faster than the network can send. When AMPS is not immediately able to send data to a client, AMPS will store the data in memory until the client is able to consume it.

If a client is oversubscribed, then it can be difficult for it to ever catch up and the memory consumed within AMPS for the client's queued messages can grow to undesirable levels. AMPS has configuration options that allow it to "offline" client messages to disk after a certain threshold of memory consumption. In some instances AMPS can even disconnect the client after the number of queued messages grows to a specified size.

To turn on the client offlining feature, set the `ClientOffline` to `enabled` and set the `ClientBufferThreshold` to the number of bytes to start offlining the client. The `ClientBufferThreshold` value specifies the buffer size for all client connected to AMPS at all times. This threshold should be set large enough that normal queries and usage will not cause a client's data to be offlined yet small enough that the number of clients multiplied by this value does not threaten the memory capacity of the host. For capacity planning, assume that every connected client will consume this `ClientBufferThreshold` value in the worst case.

For more information on how to size a deployment considering slow clients, please see the section on [Slow Clients](#).

8.2.2 Slow Client Disconnect

There are times when a client may get so far behind that it may be a better decision to disconnect it. AMPS has a feature that can disconnect slow clients that can be turned on by setting the transports `SlowClientDisconnect` property to `true` and setting `ClientOfflineThreshold` to the number of messages that need to be queued before AMPS can consider disconnecting a client. AMPS will not necessarily disconnect a client when it first crosses this threshold, but it is the point at which AMPS will start monitoring the client to see if it is making progress. If the client is not making progress towards catching

up after crossing the threshold, then AMPS will disconnect the client and free resources used by the client.



To prevent potential unbounded memory growth, by default `SlowClientDisconnect` and `ClientOffline` are enabled with `ClientBufferThreshold` set to 52MB and an offline file size limit of 1GB.

8.2.3 Protocol Details

The TCP/IP protocol requires a sender to transmit a 4-byte, unsigned, network byte-order integer that declares the size of the message payload that follows on the TCP/IP stream. This is shown in diagram [Figure 8.1](#).

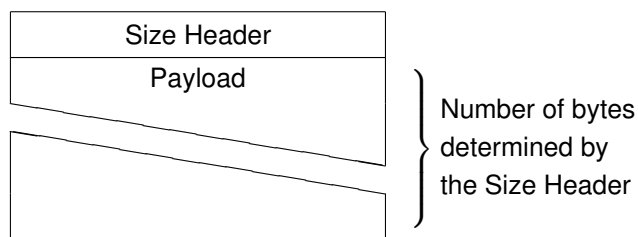


Figure 8.1: TCP/IP packet

Chapter 9

Message Types

Message communication between the client and server in AMPS is managed through the use of several different message types. This chapter will cover how AMPS uses FIX, NVFIX and XML to exchange data over the transports described in [Chapter 8](#).

9.1 Configuration

AMPS message types are configured via the `MessageTypes` section of the configuration file. Within the `MessageTypes` node of the configuration file there must be one or more `MessageType` nodes defined. Each `MessageType` requires a `Name`, `Type` and `Module`. Additionally, if `fix` or `nvfix` are specified as the `Module` then a `FieldSeparator`, `HeaderSeparator` and `MessageSeparator` must also be specified. These separators are discussed in more detail in [Section 9.2](#). [Listing 9.1](#) contains an example of an AMPS instance using `fix` as the message type provided.

```
1 <MessageTypes>
2   <MessageType>
3     <Name>fix</Name>
4     <Type>fix</Type>
5     <Module>fix</Module>
6     <FieldSeparator>1</FieldSeparator>
7     <HeaderSeparator>2</HeaderSeparator>
8     <MessageSeparator>3</MessageSeparator>
9   </MessageType>
10 </MessageTypes>
```

Listing 9.1: FIX message type configuration example.



The values used for the `FieldSeparator`, `HeaderSeparator` and `MessageSeparator` are converted to the byte values for the ASCII characters. For example, if the `FieldSeparator` is 1, then the `FieldSeparator` is `0x01`.



By default AMPS uses byte `0x01` as the default `FieldSeparator`, byte `0x02` as the `HeaderSeparator`, and byte `0x03` as the `MessageSeparator`.

9.2 FIX

AMPS supports the use of the FIX (Financial Information eXchange) protocol as one of the message type formats. FIX is based on the protocol established and maintained by FIX Protocol, Ltd. For comprehensive discussions on the FIX protocol, visit their website at <http://fixprotocol.org>.

9.2.1 FIX Message Structure in AMPS

An AMPS FIX message consists of two parts to compose a complete message -- a *message header* and a *message body*. The message header and body are separated by a `HeaderSeparator` character which is specified in the AMPS configuration file.

The contents of the message header consists of key-value pairs used to declare message attributes such as the command type and other options available to the command. In FIX keys can only be integer values and are not permitted to contain non-numeric characters.

The key-values in the header and in the message body are delimited by the `FieldSeparator` value specified in the AMPS configuration file.

An example `publish` message is listed below:

```
20005=testTopic^A20000=publish^A^B1=A^A2=25^A
```

In this example the `^A` is the ASCII 1 character which is used as the field separator for this message. The `^B` character (the ASCII 2 character) is used as the header separator between the header and the message body.

The key `20005` defines the topic to which the message will be published, which in this example is `testTopic`. The key `20000` defines the command that AMPS will issue, in this case a `publish` command. The `^B` is used to denote

the end of the message header and the beginning of the message body. In this message there are two fields being published to the AMPS topic `testTopic`, `1 = A` and `2 = 25`.

For more information on the header fields used by AMPS, please see [Appendix A](#). For more information on the commands available in an AMPS message refer to [Appendix B](#).



NVFIX is a FIX variant supported by AMPS. The difference between FIX and NVFIX is that the keys used in NVFIX can use alpha-numeric characters as opposed to the strictly integer-based keys in FIX.

9.3 SOAP

In addition to FIX and NVFIX, AMPS supports a FIX variant called FIXML. FIXML uses a SOAP envelope to deliver messages.

9.3.1 SOAP Message Structure in AMPS

A SOAP message has a similar structure to the FIX message format described in [Section 9.2](#) where the message contains a header and a message body. Instead of a key-value pairing as with FIX and NVFIX, the XML `Header` tag uses the tag-value pairing. These tag-value pairings contain attributes such as the command type and other options available to the command.

The `Body` of the SOAP message also uses the tag-value pairings to assemble the data within the SOAP message. In the example listed in [Listing 9.2](#), a client is publishing a message to the `/ett/order` topic containing the data `1=a` and `2=b`.

```

1 <?xml version="1.0" encoding="iso-8859-1"?>
2 <SOAP-ENV:Envelope>
3   <SOAP-ENV:Header>
4     <Cmd>publish</Cmd>
5     <Tpc>/ett/order</Tpc>
6   </SOAP-ENV:Header> <SOAP-ENV:Body>
7     <ett>
8       <order>
9         <1>a</1>
10        <2>b</2>
11      </order>
12    </ett>
13  </SOAP-ENV:Body>

```

```
14 </SOAP-ENV:Envelope>
```

Listing 9.2: Simple publish message in XML format.

Chapter 10

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 *Operations Guide* describes the AMPS logger configuration and the unique settings for each logging target.

10.1 Configuration

Logging within AMPS is enabled by adding a Logging section to the configuration. For example, the following would log all messages with an 'info' level or higher to the console:

```
1 <AMPSConfig>
2   ...
3   <Logging>
4     <Target>
5       <Protocol>stdout</Protocol>
6       <Level>info</Level>
7     </Target>
8   </Logging>
9   ...
10 </AMPSConfig>
```

In the example above, the `Logging` section defines a single `Target` on line 3, which is used to log all messages to the `stdout` output. Line 5 states that only messages with a log level of `info` or greater will be output to the screen.

10.2 Log Messages

An AMPS log message is composed of the following:

- Timestamp (eg: 2010-04-28T21:52:03.4766640-07:00)
- AMPS thread identifier
- Log Level (eg: info)
- Error identifier (eg: 15-0008)
- Log message

An example of a log line (it will appear on a single line within the log):

```
2011-11-23T14:49:38.3442510-08:00 [1] info: 00-0015 ←
    AMPS initialization completed (0 seconds).
```

Each log message has a unique identifier of the form `TT-NNNN` where `TT` is the component within AMPS which is reporting the message and `NNNN` the number that uniquely identifies that message within the module. Each logging target allows the direct exclusion and/or inclusion of error messages by identifier. For example, a log file which would include all messages from module `00` except for `00-0001` and `00-0004` would use the following configuration:

```
1 <Logging>
2   <Target>
3     <Protocol>stdout</Protocol>
4     <IncludeErrors>00-0002</IncludeErrors>
5     <ExcludeErrors>00-0001, 00-0004, 12-1*</ExcludeErrors>
6   </Target>
7 </Logging>
```

The above `Logging` configuration example, all log messages which are at or above the default log level of `info` will be emitted to the logging target of `stdout`. The configuration explicitly wants to see configuration messages where the error identifier matches `00-0002`. Additionally, the messages which match `00-0001`, `00-0004` will be excluded, along with any message which match the regular expression of `12-1*`.

10.3 Log Levels

AMPS has nine log levels of escalating severity. When configuring a logging target to capture messages for a specific log level, all log levels at or above that level are sent to the logging target. For example, if a logging target is configured to capture at the "error" level, then all messages at the "error", "critical", and "emergency" levels will be captured because "critical" and "emergency" are of a higher level. The following table ([Table 10.1](#)) contains a list of all the log levels within AMPS.

Table 10.1: Log Levels

Level	Description
none	no logging.
trace	all inbound/outbound data.
debug	debugging statements.
stats	statistics messages.
info	general information messages.
warning	problems that AMPS tries to correct that are often harmless.
error	events in which processing had to be aborted.
critical	events impacting major components of AMPS that if left uncorrected may cause a fatal event or message loss.
emergency	a fatal event.

Each logging target allows the specification of a `Level` attribute that will log all messages at the specified log level or with higher severity. The default `Level` is `none` which would log nothing. Optionally, each target also allows the selection of specific log levels with the `Levels` attribute. Within `Levels`, a comma separated list of levels will be additionally included.

For example, having a log of only `trace` messages may be useful for later playback, but since `trace` is at the lowest level in the severity hierarchy it would normally include all log messages. To only enable `trace` level, specify `trace` in the `Levels` setting as below:

```

1 <AMPSConfig>
2   ...
3   <Logging>
4     <Target>
5       <Protocol>gzip</Protocol>
6       <FileName>traces.log.gz</FileName>
7       <Levels>trace</Levels>
8     </Target>
9   </Logging>
10  ...
11 </AMPSConfig>

```

Logging only `trace` and `info` messages to a file is demonstrated below:

```

1 <AMPSConfig>
2   ...
3   <Logging>
4     <Target>
5       <Protocol>file</Protocol>
6       <FileName>traces-info.log</FileName>
7       <Levels>trace,info</Levels>

```

```

8   </Target>
9   </Logging>
10  ...
11 </AMPSConfig>

```

10.4 Logging to a File

To log to a file, declare a logging target with a protocol value of `file`. Beyond the standard `Level`, `Levels`, `IncludeErrors`, and `ExcludeErrors` settings available on every logging target, file targets also permit the selection of a `FileName` mask and `RotationThreshold`.

10.4.1 Selecting a FileName

The `FileName` attribute is a mask which is used to construct a directory and file name location for the log file. AMPS will resolve the file name mask using the symbols in [Table 10.2](#). For example, if a file name is masked as:

```
%Y-%m-%dT%H:%M:%S.log
```

Then AMPS would create a log file in the current working directory with a timestamp of the form: `2012-02-23T12:59:59.log`.

If a `RotationThreshold` is specified in the configuration of the same log file, the the next log file created will be named based on the current system time, not on the time that the previous log file was generated. Using the previous log file as an example, if the first rotation was to occur 10 minutes after the creation of the log file, then that file would be named `2012-02-23T13:09:59.log`.

Log files which need a monotonically increasing counter when log rotation is enabled can use the `%n` mask to provide this functionality. If a file is masked as:

```
localhost-amps-%n.log
```

Then the first instance of that file would be created in the current working directory with a name of `localhost-amps-00000.log`. After the first log rotation, a log file would be created in the same directory named `localhost-amps-00001.log`.

Log file rotation is discussed in greater detail in [Section 10.4.2](#).

Mask	Definition
%Y	Year
%m	Month
%d	Day
%H	Hour

Mask	Definition
%M	Minute
%S	Second
%n	Iterator which starts at 00000 when AMPS is first started and increments each time a <code>RotationThreshold</code> size is reached on the log file.

Table 10.2: Log filename masks

10.4.2 Log File Rotation

Log files can be 'rotated' by specifying a valid threshold in the `RotationThreshold` attribute. Values for this attribute have units of bytes unless another unit is specified as a suffix to the number. The valid unit suffixes are:

Unit Suffix	Base Unit	Examples
no suffix	bytes	"1000000" is 1 million bytes
k or K	thousands of bytes	"50k" is 50 thousand bytes
m or M	millions of bytes	"10M" is 10 million bytes
g or G	billions of bytes	"2G" is 2 billion bytes
t or T	trillions of bytes	"0.5T" is 500 billion bytes

Table 10.3: Log file rotation units



When using log rotation, if the next filename is the same as an existing file, the file will be truncated before logging continues. For example, if "amps.log" is used as the `FileName` mask and a `RotationThreshold` is specified, then the second rotation of the file will overwrite the first rotation. If it is desirable to keep all logging history, then it is recommended that either a timestamp or the %n rotation count be used within the `FileName` mask when enabling log rotation.

10.4.3 Examples

The following logging target definition would place a log file with a name constructed from the timestamp and current log rotation number in the './logs' sub-directory. The first log would have a name similar to "./logs/20121223125959-00000.log" and would store up to 2GB before creating the next log file named "./logs/201212240232-00001.log".

```

1 <AMPSConfig>
2   ...
3   <Logging>
4     <Target>
5       <Protocol>file</Protocol>
6       <Level>info</Level>
7       <FileName>./logs/%Y%m%d%H%M%S-%n.log</FileName>
8       <RotationThreshold>2G</RotationThreshold>
9     </Target>
10  </Logging>
11  ...
12 </AMPSConfig>

```

This next example will create a single log named "amps.log" which will be appended to during each logging event. If amps.log contains data when AMPS starts, that data will be preserved and new log messages will be appended to the file.

```

1 <AMPSConfig>
2   ...
3   <Logging>
4     <Target>
5       <Protocol>file</Protocol>
6       <Level>info</Level>
7       <FileName>amps.log</FileName>
8     </Target>
9   </Logging>
10  ...
11 </AMPSConfig>

```

10.5 Logging to a Compressed File

AMPS supports logging to compressed files as well. This is useful when trying to maintain a smaller logging footprint. Compressed file logging targets are the same as regular file targets except for the following:

- the `Protocol` value is 'gzip' instead of 'file';
- the log file is written with gzip compression;
- the `RotationThreshold` is metered off of the uncompressed log messages;

- makes a trade off between a small increase in CPU utilization for a potentially large savings in logging footprint.

10.5.1 Example

The following logging target definition would place a log file with a name constructed from the timestamp and current log rotation number in the `./logs` sub-directory. The first log would have a name similar to `./logs/20121223125959-0.log.gz` and would store up to 2GB of uncompressed log messages before creating the next log file named `./logs/201212240232-1.log.gz`.

```

1 <AMPSConfig>
2   ...
3   <Logging>
4     <Target>
5       <Protocol>gzip</Protocol>
6       <Level>info</Level>
7       <FileName>./logs/%Y%m%d%H%M%S-%n.log.gz</FileName>
8       <RotationThreshold>2G</RotationThreshold>
9     </Target>
10  </Logging>
11  ...
12 </AMPSConfig>

```

10.6 Logging to the Console

The console logging target instructs AMPS to log certain messages to the console. Both the standard output and standard error streams are supported. To select standard out use a `Protocol` setting of `stdout`. Likewise, for standard error use a `Protocol` of `stderr`.

10.6.1 Example

Below is an example of a console logger that logs all messages at the `'info'` or `'warning'` level to standard out and all messages at the `'error'` level or higher to standard error (which includes `'error'`, `'critical'` and `'emergency'` levels).

```

1 <AMPSConfig>
2   ...
3   <Logging>
4     <Target>
5       <Protocol>stdout</Protocol>
6       <Levels>info,warning</Levels>
7     </Target>
8     <Target>

```

```

9     <Protocol>stderr</Protocol>
10    <Level>error</Level>
11    </Target>
12  </Logging>
13    ...
14 </AMPSConfig>

```

10.7 Logging to Syslog

AMPS can also log messages to the host's syslog mechanism. To use the syslog logging target, use a `Protocol` of `syslog` in the logging target definition.

The host's syslog mechanism allows a logger to specify an identifier on the message. This identifier is set through the `Ident` property and defaults to the AMPS instance name (see [Appendix C](#) for configuration of the AMPS instance name.)

The syslog logging target can be further configured by setting the `Options` parameter to a comma-delimited list of syslog flags. Those syslog flags recognized are:

Table 10.4: Logging options available for SYSLOG configuration.

Level	Description
LOG_CONS	Write directly to system console if there is an error while sending to system logger.
LOG_NDELAY	Open the connection immediately (normally, the connection is opened when the first message is logged).
LOG_NOWAIT	No effect on Linux platforms.
LOG_ODELAY	The converse of LOG_NDELAY; opening of the connection is delayed until <code>syslog()</code> is called. (This is the default, and need not be specified.)
LOG_PERROR	Print to standard error as well.
LOG_PID	Include PID with each message.



AMPS already includes the process identifier (PID) with every message it logs, however, it is a good practice to set the `LOG_PID` flag so that downstream syslog analysis tools will find the PID where they expect it.

The `Facility` parameter can be used to set the syslog 'facility'. Valid options are: `LOG_USER` (the default), `LOG_LOCAL0`, `LOG_LOCAL1`, `LOG_LOCAL2`, `LOG_LOCAL3`, `LOG_LOCAL4`, `LOG_LOCAL5`, `LOG_LOCAL6`, or `LOG_LOCAL7`.

Finally, AMPS and the syslog do not have a perfect mapping between their respective log severity levels. AMPS uses the following table to convert the AMPS log level into one appropriate for the syslog:

Table 10.5: Comparison of AMPS log severity to Syslog severity.

AMPS Severity	Syslog Severity
none	LOG_DEBUG
trace	LOG_DEBUG
debug	LOG_DEBUG
stats	LOG_INFO
info	LOG_INFO
warning	LOG_WARNING
error	LOG_ERR
critical	LOG_CRIT
emergency	LOG_EMERG

10.7.1 Example

Below is an example of a syslog logging target that logs all messages at the 'critical' severity level or higher and additionally the log messages matching 30-0001 to the syslog.

```

1 <AMPSConfig>
2   ...
3   <Logging>
4     <Target>
5       <Protocol>syslog</Protocol>
6       <Level>critical</Level>
7       <IncludeErrors>30-0000</IncludeErrors>
8       <Ident>\amps dma</Ident>
9       <Options>LOG_CONS,LOG_NDELAY,LOG_PID</Options>
10      <Facility>LOG_USER</Facility>
11    </Target>
12  </Logging>
13  ...
14 </AMPSConfig>

```

10.8 Error Categories

In the AMPS log messages, the error identifier consists of an error category, followed by a hyphen, followed by an error identifier. The error categories cover the different modules and features of AMPS, and can be helpful in diagnostics

and troubleshooting by providing some context about where a message is being logged from. A list of the error categories found in AMPS are listed in [Table 10.6](#)

Table 10.6: AMPS Error Categories.

AMPS Severity	Syslog Severity
00	AMPS Startup
01	General
02	Message Processing
03	Expiration
04	Publish Engine
05	Statistics
06	Metadata
07	Client
08	Regex
09	ID Generator
0A	Diff Merge
0C	View
0D	Message Data Cache
0E	Topic Replicas
0F	Message Processor Manager
11	Connectivity
12	Trace In - for inbound messages
13	Datasource
14	Subscription Manager
15	SOW
16	Query
17	Trace Out - for outbound messages
18	Parser
19	Administrative Console
1A	Evaluation Engine
1B	SQLite
1C	Meta Data Manager
1D	Transaction Log Manager
1E	Replication
1F	Client Session
20	Global Heartbeat
21	Transaction Replay
22	TX Completion
23	Bookmark Subscription
24	Thread Monitor
FF	Shutdown

10.9 Error discovery with amperr

In the `$AMPSDIR/bin` directory is the `amperr` utility. Running this utility is useful for getting detailed information and messages about specific AMPS errors observed in the log files.



AMPS errors follow the format of `XX-YYYY` where `XX` is a grouping associated with the message and `YYYY` is a specific error within that grouping. For example if `XX` is `12`, this is the group for the logging of inbound messages.

The `amperr` utility can be used to lookup the detailed description and - where known - an action to assist in recovery or mitigation of the error. For example, if the error `05-0021` was found in the log, more information can be found about this error by typing

```
./amperr 05-0021
```

in the command line. To which `amperr` will return the following information with a `DESCRIPTION` and a recommended `ACTION` as seen below:

```
./amperr 05-0021 AMPS Message 05-0021 [level
= critical]

DESCRIPTION : AMPS was unable to service the statistics
query over the HTTP administration port.

ACTION      : This problem could be temporary, but if
it does not go away after 30 seconds, then it may be
indicative of a more serious issue with a hung AMPS
message processor. If you are also seeing "stuck"
messages in the log, please restart AMPS to clear the
issue and report the problem to AMPS support.

Found 1 error matching '05-0021'.
```

Chapter 11

Event Topics

AMPS publishes specific events to internal topics that begin with the `/AMPS/` prefix, which is reserved for AMPS only. For example, all client connectivity events are published to the internal `/AMPS/ClientStatus` topic. This allows all clients to monitor for events that may be of interest.



Event topic messages can be subscribed with content filters like any other topic within AMPS.

11.1 Client Status

The AMPS engine will publish client status events to the internal `/AMPS/ClientStatus` topic whenever a client issues a `logon` command, `disconnects` or issues a `subscribe` or `unsubscribe` command. In addition, upon a `disconnect`, a client status message will be published for each subscription that the client had registered at the time of the `disconnect`. This mechanism allows any client to monitor what other clients are doing and is especially useful for publishers.

To help identify clients, it is recommended that clients send a `logon` command to the AMPS engine and specify a client name. This client name is used to identify a client and does not have to be unique as the AMPS engine only includes it for identification within client status event messages, logging output, and information on clients within the monitoring console.

Each message published to the client status topic will contain an `Event` and a `ClientName`. For `subscribe` and `unsubscribe` events, the message will contain `Topic`, `Filter` and `SubId`. The `Filter` will only be sent if it is not null.

When the AMPS engine is configured as `xml`, the client status message published to the `/AMPS/ClientStatus` will contain a SOAP body with a `ClientStatus` section as follows:

```

1 <?xml version="1.0" encoding="iso-8859-1"?>
2 <SOAP-ENV:Envelope>
3   <SOAP-ENV:Header>
4     <Cmd>publish</Cmd>
5     <TxmTm>20090106-23:24:40-0500</TxmTm>
6     <Tpc>/AMPS/ClientStatus</Tpc>
7     <MsgId>MAMPS-55</MsgId>
8     <SubId>SAMPS-1233578540_1</SubId>
9   </SOAP-ENV:Header>
10  <SOAP-ENV:Body>
11    <ClientStatus>
12      <Event>subscribe</Event>
13      <ClientName>test_client</ClientName>
14      <Topic>order</Topic>
15      <Filter> (/FIXML/Order/Instrmt/@Sym = 'IBM')</Filter>
16      <SubId>SAMPS-1233578540_10</SubId>
17    </ClientStatus>
18  </SOAP-ENV:Body>
19 </SOAP-ENV:Envelope>

```

Table 11.1 defines the header fields which may be returned as part of the subscription messages to the `/AMPS/ClientStatus` topic.

FIX	XML	Definition
20065	Timestamp	Timestamp in which AMPS sent the message.
20066	Event	Command executed by the client.
20067	ClientName	Client Name.
20068	Tpc	SOW Topic.
20069	Filter	Filter (if applicable).
20070	SubId	Subscription ID (if applicable).
20071	ConnName	Internal AMPS connection name.

Table 11.1: `/AMPS/ClientStatus` Format Fields

11.2 SOW Statistics

AMPS can publish SOW statistics for each SOW topic which has been configured. To enable this functionality, specify the `SOWStatsInterval` in the configuration file. The value provided in `SOWStatsInterval` is the time between updates to the `/AMPS/SOWStats` topic.

For example, the following would be a configuration that would publish internal `/AMPS/SOWStats` event messages every 5 seconds.

```

1 <AMPSConfig>
2   ...
3   <SOWStatsInterval>5s</SOWStatsInterval>
4   ...
5 </AMPSConfig>

```

When the AMPS engine is configured as `xml`, the SOW status message published to the `/AMPS/SOWStats` topic will contain a SOAP body with a `SOWStats` section as follows:

```

1 <?xml version="1.0" encoding="iso-8859-1"?>
2 <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
3   <SOAP-ENV:Header>
4     <Cmd>publish</Cmd>
5     <TxmTm>2010-09-08T17:49:06.9439120Z</TxmTm>
6     <Tpc>/AMPS/SOWStats</Tpc>
7     <SowKey>18446744073709551615</SowKey>
8     <MsgId>AMPS-10548998</MsgId>
9     <SubIds>SAMPS-1283968028_2</SubIds>
10  </SOAP-ENV:Header>
11  <SOAP-ENV:Body>
12    <SOWStats>
13      <Timestamp>2010-09-08T17:49:06.9439120Z</Timestamp>
14      <Topic>MyTopic</Topic>
15      <Records>10485760</Records>
16    </SOWStats>
17  </SOAP-ENV:Body>
18 </SOAP-ENV:Envelope>

```

In the `SOWStats` message, the `Timestamp` field includes the time the event was generated, `Topic` includes the topic, and `Records` includes the number of records.

Table 11.2 defines the header fields which may be returned as part of the subscription messages to the `/AMPS/SOWStats` topic.

FIX	XML	Definition
20065	<code>Timestamp</code>	Timestamp in which AMPS sent the message.
20066	<code>Topic</code>	Topic that statistics are being reported on.
20067	<code>Records</code>	Number of records in the SOW topic.

Table 11.2: Client Status FIX Format Fields

11.3 Persisting Event Topic Data

By default, AMPS event topics are not persisted to the SOW. However, because AMPS event topic messages are treated the same as all other messages, the event topics can be persisted to the SOW. Providing a topic definition with the appropriate `Key` definition can resolve that issue by instructing AMPS to persist the messages. For example, to persist the last `/AMPS/SOWStats` message for each topic in both `fix` and `xml` format, the following `TopicDefinition` sections could be added to the AMPS configuration file:

```
1 <TopicMetaData>
2   <TopicDefinition>
3     <FileName>./sow/sowstats.fix.sow</FileName>
4     <Topic>/AMPS/SOWStats</Topic>
5     <Key>/20066</Key>
6     <MessageType>fix</MessageType>
7   </TopicDefinition>
8   <TopicDefinition>
9     <FileName>./sow/sowstats.xml.sow
10    <Topic>/AMPS/SOWStats</Topic>
11    <Key>/Topic</Key>
12    <MessageType>xml</MessageType>
13  </TopicDefinition>
14 </TopicMetaData>
```

Every time an update occurs, AMPS will persist the `/AMPS/SOWStats` message and it will be stored twice, once to the `fix` SOW topic, and once to the `xml` SOW topic. Each update to the respective SOW topic will overwrite the record with the same `Topic` or `20066` tag value. Doing this allows clients to now query the `SOWStats` topic instead of actively listening to live updates.

Chapter 12

Message Acknowledgment

AMPS enables a client which sends commands to AMPS to request the status of those commands at various check points throughout the message processing sequence. These status updates are handled in the form of `ack` messages. Each command supports a different set of acknowledgment messages, and each of those messages will be discussed in this chapter.

12.1 Subscription Acknowledgment Messages

AMPS supports a variety of acknowledgment messages which are set in the `AckTyp` header field of the message. The supported `AckTyp` are listed here:

Table 12.1: Acknowledgment messages supported by AMPS.

Ack Type	Definition
<code>none</code>	Do not to return an ack (default).
<code>received</code>	Return an ack as soon as it receives the message.
<code>persisted</code>	Return an ack after a publish message has been successfully persisted to the SOW cache, transaction log, and all synchronous downstream replication destinations. This ack type only applies to publish messages. (For more information please see Topic Replicas .)
<code>processed</code>	Return an ack after it has successfully started processing the message.
<code>completed</code>	Return an ack after a message has been successfully processed. The exact meaning is command dependent.

Continued on next page

Table 12.1 -- continued from previous page

Ack Type	Definition
stats	Return a statistics ack which summarizes statistics of the executed command.

Subscribe and unsubscribe command messages only support *received* and *processed* acknowledgment types. If a *subscribe* command succeeds, then the acknowledgment message *Status* header field will indicate success. If the *Status* is set to *failure* then the message will additionally contain a *Reason* which will describe why the failure occurred. Example *Reason* returns would be *duplicate* if a duplicate message is published to AMPS, or *bad filter* if a query contains a filter that can not be parsed.

If the subscriber did not provide a subscription ID as part of the subscription request, then the AMPS engine will generate a unique subscription ID that will be returned in the *SubId* header field.

The following is an example ack message in response to a subscribe command:

```

1 <?xml version="1.0" encoding="iso-8859-1"?>
2 <SOAP-ENV:Envelope>
3   <SOAP-ENV:Header>
4     <Cmd>ack</Cmd>
5     <TxmTm>20080210-17:16:46.066-0500</TxmTm>
6     <CmdId>1</CmdId>
7     <AckTyp>processed</AckTyp>
8     <Status>success</Status>
9     <MsgId>MAMPS-330</MsgId>
10    <SubId>SAMPS-1202681806_1</SubId>
11  </SOAP-ENV:Header>
12  <SOAP-ENV:Body>
13  </SOAP-ENV:Body>
14 </SOAP-ENV:Envelope>

```

12.1.1 AMPS Commands and Supported Acknowledgment Types

Each of the *Cmd* messages sent to AMPS can have a different *AckType* returned to the client. A summary of the acknowledgment messages available to the specific commands is summarized in [Table 12.2](#).

Table 12.2: Commands and supported acknowledgment types.

Command	none	received	persisted	processed	completed	stats
delta_publish	•	•	•	•		
delta_subscribe	•		•	•		
logon	•	•		•		
publish	•	•	•	•		
sow_and_delta_subscribe	•	•		•	•	•
sow_and_subscribe	•	•		•	•	•
sow_delete	•	•	•	•	•	•
sow	•	•		•	•	
subscribe	•	•		•	•	
unsubscribe	•	•		•		

Chapter 13

Topic Replicas

To further reduce network bandwidth consumption, AMPS supports a feature called "topic replicas". A topic replica is a copy of one topic into another with the ability to control the replication interval.

To better see the value in a topic replica, 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 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 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.



A topic replica is distinct from a high-availability replication destination in that the goal of the topic replica is to down-sample the data to make the data flow to a client manageable. A high availability replica is implemented with the goal of data redundancy and high uptime. The [High Availability](#) has more detail about high availability replication.

13.1 Configuration

Configuration of a replica topic involves creation of a `ReplicaDefinition` section in the AMPS configuration file. Here is an example of a regular SOW topic named `FastPublishTopic` and its replica definition, `replica_FastPublishTopic`. In this example, the replication interval is set at 5s (five seconds). For more information about how units work in AMPS configuration see [Using Units in the Configuration](#).

```

1 <TopicDefinition>
2   <Topic>FastPublishTopic</Topic>
3   <FileName>./sow/%n.sow</FileName>
4   <MessageType>fix</MessageType>
5   <Key>/1</Key>
6 </TopicDefinition>
7
8 <ReplicaDefinition>
9   <Topic>replica_FastPublishTopic</Topic>
10  <FileName>./sow/%n.sow</FileName>
11  <MessageType>fix</MessageType>
12  <UnderlyingTopic>FastPublishTopic</UnderlyingTopic>
13  <Interval>5s</Interval>
14 </ReplicaDefinition>

```



Topic Replicas require underlying SOW topics. See [State of the World \(SOW\)](#) for more information on creating and configuring SOW topics.

The configuration parameters available when defining a topic replica are included in the [13.1](#). Each parameter should be included within a `ReplicaDefinition` section.

Parameter	Description
Topic	String used to define the replica topic name.
UnderlyingTopic	String used to define the SOW topic which provides updates to the replica.
FileName	The file location to store the topic replica data.
MessageType	The message format used to store the data from the underlying topic.
Interval	Default of 5 second interval between replication event. See Section C.1.2 for more information on intervals.

Table 13.1: Topic Replica Configuration Parameters



A replica topic can only be created on a SOW topic that has been defined in the AMPS configuration. Ad-hoc topics can not have replicas.



It is a good idea to name your replica topic something similar to the underlying topic. For example, if the underlying topic is named `ORDER_STATE` then a good name for the replica is something like `ORDER_STATE-REPLICA` or `ORDER_STATE-R`.



Messages can not be published to a topic replica. Messages published to the underlying topic will be published to subscribers of the topic replica.

Chapter 14

View Topics

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.

14.1 Example

For a potential usage scenario, imagine the topic `ORDERS` which includes the following FIX message schema:

Table 14.1: `ORDERS` table identifiers

FIX Tag	Description
37	unique order identifier
55	symbol
109	unique client identifier
14	currently executed shares for the chain of orders
6	average price for the chain of orders

This topic includes information on the current state of executed orders, but may not include all the information we want updated in real-time. For example, we may want to monitor the total value of all orders executed by a client at any moment. If `ORDERS` was a SQL Table within an RDBMS the "view" we would want to create would be similar to:

```
CREATE VIEW TOTAL_VALUE AS
SELECT 109, SUM(14*6) AS 71406
FROM ORDERS
GROUP BY 109
```

As defined above, the `TOTAL_VALUE` view would only have two fields:

1. 109 : the client identifier
2. 71406: the summation of current order values by client

Views in AMPS are specified in the AMPS configuration file in `ViewDefinition` section, which itself must be defined in the `TopicMetaData` section. The example above would be defined as:

```

1 <TopicMetaData>
2 <TopicDefinition>
3   <Topic>ORDERS</Topic>
4   <MessageType>fix</MessageType>
5   <Key>/37</Key>
6 </TopicDefinition>
7 <ViewDefinition>
8   <Topic>TOTAL_VALUE</Topic>
9   <UnderlyingTopic>ORDERS</UnderlyingTopic>
10  <FileName>./views/totalValue.view</FileName>
11  <MessageType>fix</MessageType>
12  <Projection>
13    <Field>/109</Field>
14    <Field>SUM(/14 * /6) AS /71406</Field>
15  </Projection>
16  <Grouping>
17    <Field>/109</Field>
18  </Grouping>
19 </ViewDefinition>
20 </TopicMetaData>

```



Views require an underlying SOW topic. See [State of the World \(SOW\)](#) for more information on creating and configuring SOW topics.

The `Topic` element is set to the new topic that is being defined. This `Topic` value will be the topic that can be used by clients to subscribe for future updates or perform SOW queries against.

The `UnderlyingTopic` is the topic that the view operates on. That is, the `UnderlyingTopic` is where the view gets its data from. All XPath references within the `Projection` fields are references to values within this underlying topic (unless they appear on the right-hand side of the `AS` keyword.)

The `Projection` section is a list of 1 or more `Field`'s that define what the view will contain. The expressions can contain either a raw XPath value, as in `/109` above, which is a straight copy of the value found in the underlying topic into the view topic using the same target XPath. If we had wanted to translate the 109 tag into a different tag, such as 10109, then we could have used the `AS`

keyword to do the translation as in `/109 AS /10109`. As is the case with SQL VIEWS, any straight references must also be a part of the grouping constraint.



Any straight reference that is not used within an aggregate function must also be a part of the grouping field list.



Usually an unexpected 0 (zero) in aggregate field within a view means that either the value is zero or NaN. AMPS defaults to using 0 instead of NaN. Any numeric aggregate function will result in a NaN result when any of the fields that are involved in the aggregation are not a number.

Finally, the `Grouping` section is a list of 1 or more `Field`'s that define how the records in the underlying topic will be grouped. In this example, we grouped by the tag holding the client identifier. However, we could have easily made this the `Symbol` tag `/55`.

The following table lists the available aggregate functions:

Table 14.2: Aggregate functions.

Function	Description
AVG	Average over an expression
COUNT	Count of values in an expression
SUM	Summation over an expression

Null values are not included in aggregate expressions with AMPS nor ANSI SQL. `COUNT` will only count non-null values, `SUM` will only add non-null values, and `AVG` will only average non-null values.



Views are only currently supported for the `fix` and `nvfix` message type.

In the below example, we want to count the number of orders by client that have a value greater than 1,000,000:

```

1 <ViewDefinition>
2 <Topic>NUMBER_OF_ORDERS_OVER_ONEMILL</Topic>
```

```

3 <UnderlyingTopic>ORDERS</UnderlyingTopic>
4 <Projection>
5   <Field>/109</Field>
6   <Field><![CDATA[SUM(IF (/14 * /6 > 1000000)) AS ↵
   /90010]]></Field>
7   <Field>SUM(IF (/14 * /6 &gt; 1000000)) AS /90011</↵
   Field>
8 </Projection>
9 <Grouping>
10  <Field>/109</Field>
11 </Grouping>
12 <FileName>
13  ./views/numOfOrdersOverOneMil.view
14 </FileName>
15 <MessageType>fix</MessageType>
16 </ViewDefinition>

```

Notice that the /90010 and /90011 will contain the same value, however /90010 was defined using an XML CDATA block and /90011 was defined using the XML > entity reference.



Since the AMPS configuration is XML, special characters in projection expressions have to be escaped with XML entity references or wrapped in a CDATA section.

Updates to an underlying topic can cause twice as many updates to a downstream view, this can create stress on downstream clients subscribed to the view. If the underlying topic has frequent updates to the same records and/or a real-time view is not required, as in a GUI, then a replica of the topic may be a good solution to reduce the frequency of the updates and conserve bandwidth. For more on topic replicas please see [Topic Replicas](#).

Chapter 15

Message Expiration

A default configuration of AMPS will store all messages which match a SOW topic indefinitely. For scenarios where message persistence needs to be limited in duration, AMPS provides the ability to set a time limit on the lifespan of SOW topic messages. This limit on duration is known as message expiration and can be thought of as a “Time to Live” feature for SOW topic messages.

15.1 Usage

There are two ways message expiration time can be set. The first, each topic can specify a default lifespan for all messages stored for that SOW topic. The second, each message can provide an expiration as part of the message header.

The expiration for a given SOW topic message is first determined based on the message expiration specified in the message header. If a message has no expiration specified in the header, then the message will inherit the expiration setting for the topic expiration. If there is no message expiration and no topic expiration, then it is implicit that a SOW topic message will not expire.

15.1.1 Topic Expiration

AMPS configuration supports the ability to specify a default message expiration for all messages in a single SOW topic. Below is an example of a configuration section for a SOW topic definition with an expiration. The [State of the World \(SOW\)](#) chapter has more detail on how to configure the SOW topic.

```
1 <TopicMetaData>
2 <TopicDefinition>
3   <FileName>sow/%n.sow</FileName>
4   <Topic>ORDERS</Topic>
```

```

5   <Expiration>30s</Expiration>
6   <Key>/55</Key>
7   <Key>/109</Key>
8   <MessageType>fix</MessageType>
9   <RecordSize>512</RecordSize>
10  </TopicDefinition>
11  </TopicMetaData>

```

15.1.2 Message Expiration

Individual messages have the ability to specify an expiration for a published message. Below is an example of an XML message which is publishing an Order, and has an expiration set for 20 seconds.

```

1  <?xml version="1.0" encoding="iso-8859-1"?>
2  <SOAP-ENV:Envelope>
3    <SOAP-ENV:Header>
4      <Cmd>publish</Cmd>
5      <TxmTm>20061201-17:29:12.000-0500</TxmTm>
6      <Expn>20</Expn>
7      <Tpc>order</Tpc>
8    </SOAP-ENV:Header>
9    <SOAP-ENV:Body>
10     <FIXML>
11       <Order Side="2" Px="32.00"><Instrmt Sym="MSFT"/<↵
12         ><OrdQty Qty="100"/></Order>
13     </FIXML>
14   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

15.2 Example Message Lifecycle

Assuming that a SOW topic message has either the message or topic expirations set, then the message will be placed into the SOW with that expiration time. If the message is not updated (or gets deleted) before it expires, then the message will be deleted from the SOW.

SOW topic messages can receive updates before expiration. When a message is updated, the message's expiration lifespan is reset. For example, a message is first published to a SOW topic, and it is set to expire in 45 seconds. The message is updated 15 seconds after publication of the initial message, and the original message will have the expiration reset to a new 45 second lifespan. This process can continue for the entire lifespan of the message, causing a new 45 second lifespan renewal for the message with every update.

If the message expires, then that message is deleted from the SOW topic. This event will trigger delete processing to be executed for the message, similar

to the process of executing a `sow_delete` command on a message stored in a SOW topic. See [sow_delete](#) for more information on how SOW message deletion works.

15.2.1 Recovery and Expiration

One scenario in the message expiration lifecycle is where a message has an expiration set, but the AMPS instance gets shut down during the lifetime of the message.

To handle such a scenario, AMPS ensures that message expiration time is requested relative to the update time of the original message; however, the time of expiration is calculated as an absolute time. Therefore, if the AMPS instance is shutdown, then upon recovery the engine will check to see which messages have expired since the occurrence of the shutdown. Any expired messages will be deleted as soon as possible.

Chapter 16

Out of Focus Message Processing (OOF)

When part of a SOW query changes such that it no longer meets the criteria that previously caused its delivery, AMPS can notify the subscriber via an *out-of-focus* (OOF) message. OOF processing is the AMPS method of notifying a client that a new message has caused a SOW record's state to change, thus informing the client that a message which previously matched their filter criteria no longer matches or was deleted.

16.1 Usage

Consider the following scenario where AMPS is configured with the following SOW key for the buyer topic:

```
1 <TopicDefinition>
2   <Topic>buyer</Topic>
3   <MessageType>xml</MessageType>
4   <Key>/buyer/id</Key>
5 </TopicDefinition>
```

Listing 16.1: Topic Configuration

When the following message is published, it is persisted in the SOW topic:

```
1 <buyer><id>100</id><loc>NY</loc></buyer>
```

If a client issues a `sow` or a `sow_and_subscribe` request with `SendOOF` enabled, topic `buyer` and filter `/buyer/loc="NY"` then the client will be sent the messages as part of the SOW query result.

Subsequently when the following message is published to update the `loc` tag to `LN` from `NY`,

```
1 <buyer><id>100</id><loc>LN</loc></buyer>
```

the original message in the SOW cache will be updated and the client will be holding a message that is no longer in the SOW cache. The AMPS engine sends an OOF message to let these clients know that the message that they hold is no longer in the SOW cache. The following is an example of what's returned:

```
1 <?xml version="1.0" encoding="iso-8859-1"?>
2 <SOAP-ENV:Envelope>
3   <SOAP-ENV:Header>
4     <Reason>match</Reason>
5     <Tpc>buyer</Tpc>
6     <Cmd>oof</Cmd>
7     <MsgTyp>xml</MsgTyp>
8     <SowKey>6387219447538349146</SowKey>
9     <SubIds>SAMPS-1214725701_1</SubIds>
10  </SOAP-ENV:Header>
11  <SOAP-ENV:Body>
12    <client><id>100</id><loc>LN</loc></client>
13  </SOAP-ENV:Body>
14 </SOAP-ENV:Envelope>
```

An easy way to think about the above is to consider what would happen if the client re-issued the original sow request after the above message was published. The `/client/loc="NY"` expression no longer matches the message in the SOW cache and as a result, this message would not be returned.

When AMPS returns an OOF message, the data contained in the body of the message represents the updated state of the OOF message. This will allow the client to make a determination as to how to handle the data, be it to remove the data from the client view or to change their query to broaden the filter thresholds.

In the case of a `delta_publish` message, the merged record will be returned in the body of the message.

For deletions and expirations the body of the message returned is the original message since it would otherwise be empty. The `Reason` contained in the message header will state whether the message was `deleted` or `expired`.

16.2 Example

To help reinforce the concept of OOF messages, and how OOF messaging can be used in AMPS, consider a scenario where there is a GUI application whose requirement is to display all open orders of a client. There are several possible solutions of ensuring the GUI client data is constantly updated as information

changes, some of which are examined below, however the goal of this section is to build up a `sow_and_subscribe` message to demonstrate the power that adding a `SendOOF` header adds to an AMPS query.

16.2.1 `sow_and_subscribe` With Client Filtering

The first approach is to send an `sow_and_subscribe` message using the following filter:

```
Topic:orders;Filter:/Client = "Adam"
```

Once AMPS has completed the `sow` portion of this call by sending all matching messages from the `orders` SOW topic, AMPS will then place a subscription where by all future messages which match the filter will be sent to the subscribing GUI client.

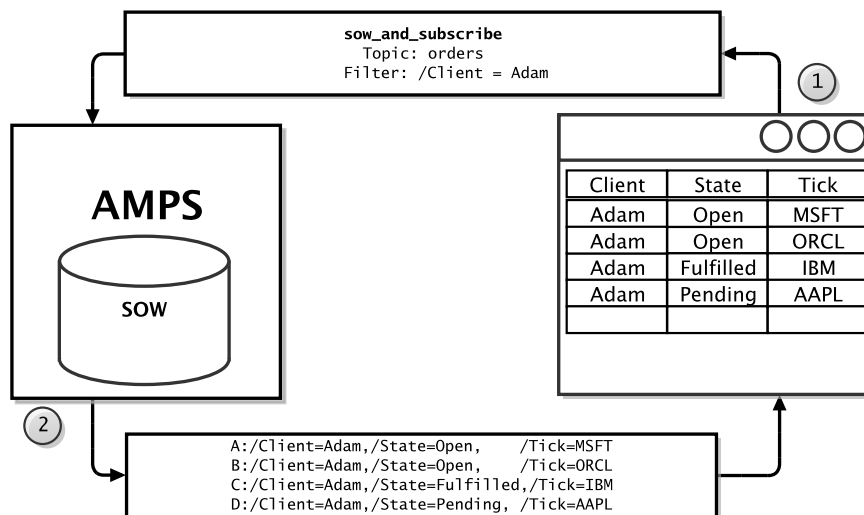


Figure 16.1: `sow_and_subscribe` example.

As the messages come in, the GUI client will be responsible for determining the state of the order by examining the `State` field and determining if the state is equal to "Open" or not and then deciding how to update the GUI based on the information returned.

This approach puts the burden of work on the GUI and in a high volume environment has the potential to make the client GUI unresponsive due to the potential load that this filtering can place on a CPU. If a client GUI becomes unresponsive, AMPS will queue the messages to ensure that the client is given the opportunity to catch up. The specifics of how AMPS handles slow clients is covered in [Slow Clients](#).

16.2.2 sow_and_subscribe With Topic Filter

The next step is to add an additional 'AND' clause to the filter. In this scenario we can let AMPS do the filtering work that was previously handled on the client. This is accomplished by modifying our original `sow_and_subscribe` using the following filter:

```
Topic:orders;Filter:/Client = "Adam" and /State = "Open"
```

Similar to the above case, this `sow_and_subscribe` will first send all messages from the `orders` SOW topic which have a `Client` field matching "Adam" and a `State` field matching "Open." Once all of the SOW topic messages have been sent to the client, the subscription will make sure all future messages which match the filter are sent to the client.

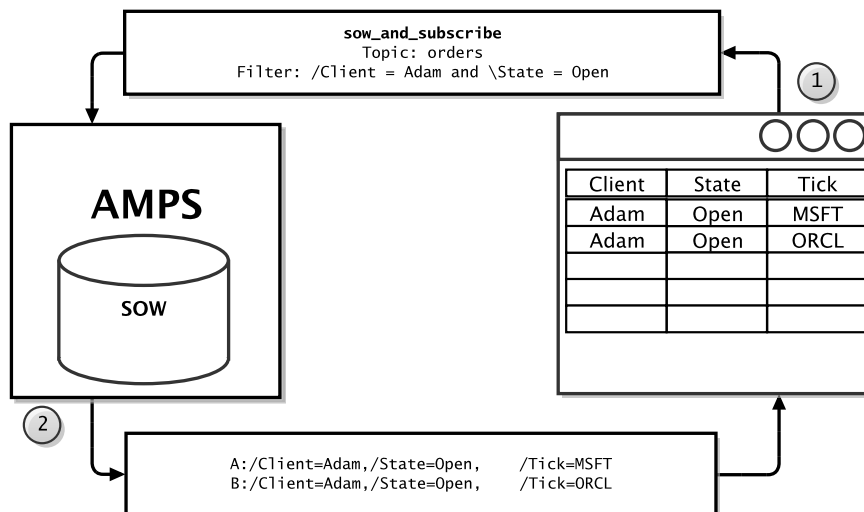


Figure 16.2: `sow_and_subscribe` with State filter.

There is a less obvious issue with this approach to maintaining the client state. The problem with this solution is that initially it will yield all of the open orders for client "Adam", however this scenario is unable to stay in sync with the server. For example, when the order for Adam is filled the `State` changes to `State=Filled`. This means a inside AMPS the order on the client will no longer match the initial filter criteria, resulting in out-of-sync records being maintained by the client. Since the client is not subscribed to messages with a `State` of "Filled" the GUI client would never be updated to reflect this change.

16.2.3 sow_and_subscribe With OOF Processing

The final solution is to implement the same `sow_and_subscribe` query which was used in the first scenario, but include `SendOOF` so the client will receive

OOF messages

```
Topic:orders;Filter:/Client = "Adam" and /State = <-
  "Open"; SndOOF=true
```

AMPS will respond immediately with the query results, in a similar manner to a `sow_and_subscribe` (Figure 16.3) command.

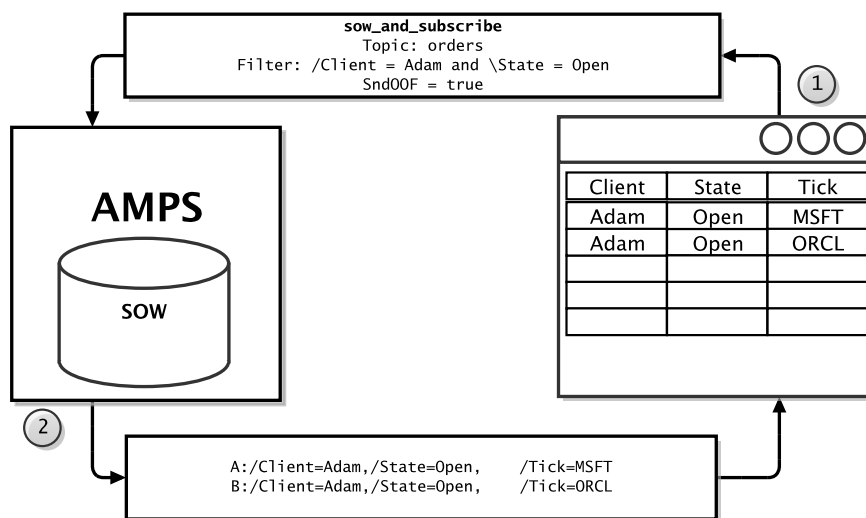


Figure 16.3: `sow_and_subscribe` with `oof` enabled

The advantage to this approach is that for all future messages, if the same `Open` order is updated such that it's status is no longer `Open`, AMPS will send the client an OOF message specifying that the record which previously matched the filter criteria has fallen out of focus. AMPS will not send any further information about the message unless another incoming AMPS message causes that message to come back into focus.

In Figure 16.4 the Publisher publishes a message stating that Adam's order for MSFT has been fulfilled. When AMPS processes this message, it will notify the GUI client with an OOF message that the original record no longer matches the filter criteria. The OOF command will include a `Reason` field with it in the message header defining the reason for the message to lose out of focus. In this case the `Reason` field will state `match` since the record no longer matches the filter.

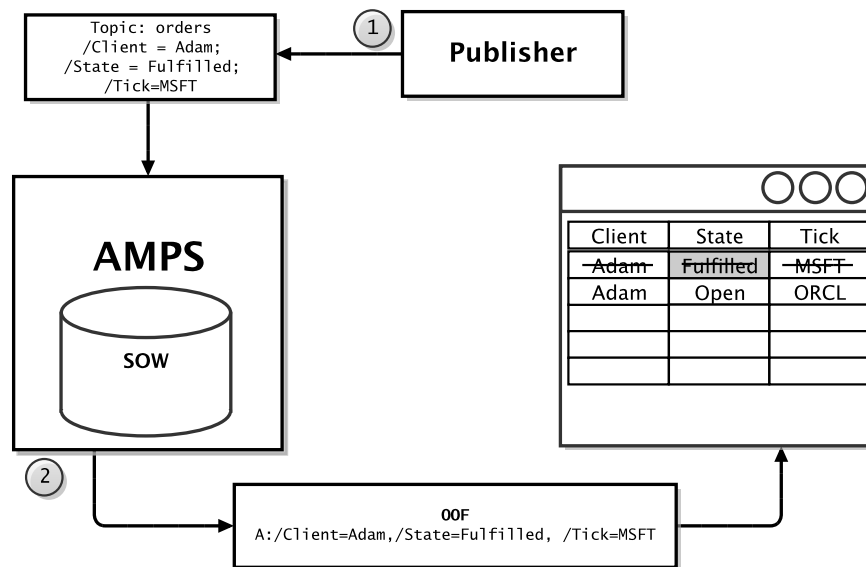


Figure 16.4: OOF message

AMPS will also send `OOF` messages when a message is deleted or has expired from the SOW topic.

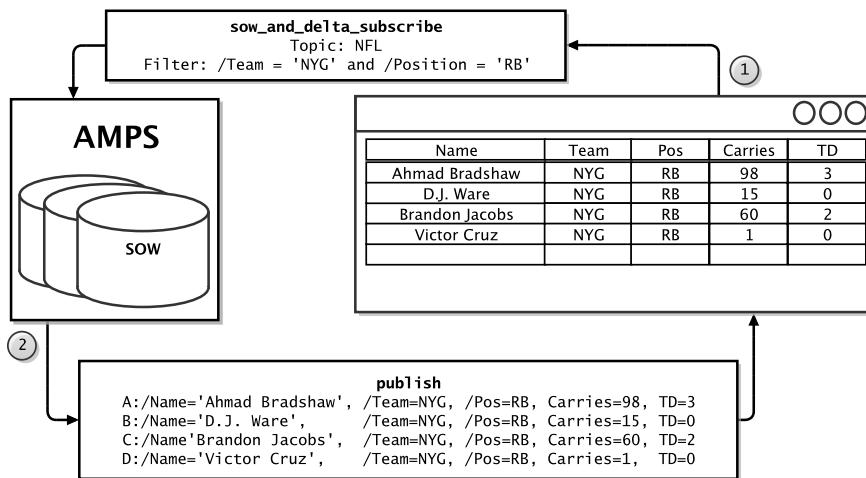
The power of the `OOF` message is when a client application wants to have a local cache that is a subset of the SOW. This is best managed by first issuing a query filter `sow_and_subscribe` which populates the GUI, and enabling `SendOOF` to know when those records which originally matched no longer do, and can be removed from the client.

16.3 Another Example

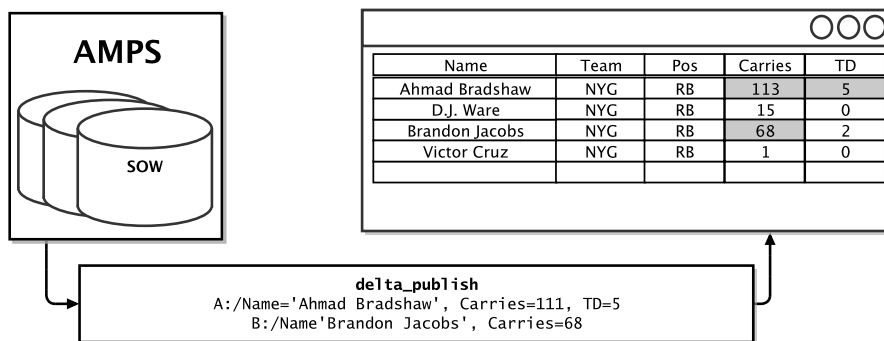
Below is an example describing how AMPS works with an `OOF` enabled `sow_and_delta_subscribe` command, however this example will examine a scenario where AMPS is used to filter messages to a client maintaining statistics regarding the roster of an NFL football team.

In this scenario, there is a data source which publishes statistics and player transactions for all professional sports to an AMPS instance. Within this AMPS instance there is a topic for each individual sports league (i.e., an NFL topic for football, an MLB topic for baseball and an NHL topic for hockey).

A client who is interested in the viewing the statistics and transaction for all of the running backs for the New York Giants football team would be able to send a `sow_and_delta_subscribe` command to the AMPS NFL topic. AMPS would then return the full records of all players who matched the query as seen in [Figure 16.5](#)

Figure 16.5: Initial `sow_and_delta_subscribe`

From this point forward, every Sunday (and occasionally on Monday) the client will receive updates about the players' stats which are subscribed to by the `sow_and_delta_subscribe`. As an example assume Ahmad Bradshaw carries the ball 13 times and scores 2 touchdowns, and Brandon Jacobs carries the ball 8 times. The `delta_publish` message would look something like the message passed to the client in Figure 16.6.

Figure 16.6: `delta_publish` message after a game

As demonstrated, AMPS will only send the statistics which were affected by the previous week's game. This minimizes the amount of data which must be sent over the network versus re-sending the entire `publish` record for both players with statistics to be updated. From the `delta_subscribe` message data we can infer D.J. Ware and Victor Cruz saw no playing time, or they did play and did not get any carries. Likewise, notice that Brandon Jacob's touchdown statistic didn't change and therefore wasn't resent to the client.

The following week, the New York Giants broker a deal with the Cleveland Browns to trade Brandon Jacobs for Peyton Hillis - both running backs. This

trade would signal two events to AMPS:

1. Peyton Hillis now matches the previous filter (he has become a running back for the New York Giants) and needs to be added to the client's results via a `publish` message.
2. Brandon Jacobs is no longer with the New York Giants, thus no longer matches the filter criteria and needs to have an `oof` message sent to the client so the client knows to remove the data from its table.

Both of these events are visualized in [Figure 16.7](#)

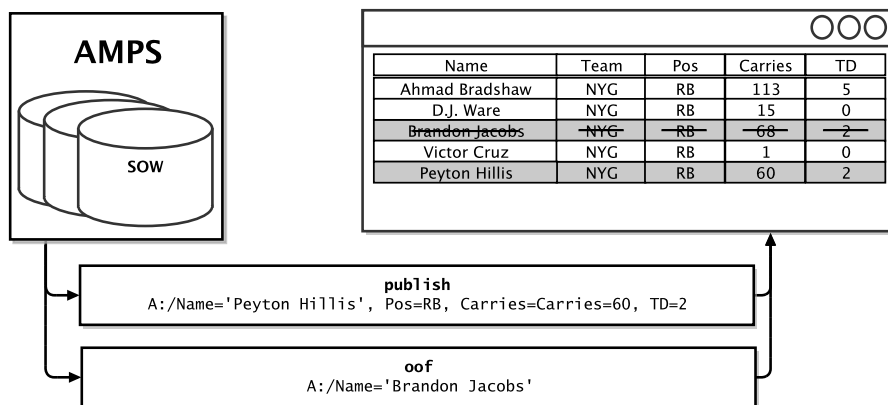


Figure 16.7: `publish` and `oof` after a trade

In a `sow_and_delta_subscribe`, when a new record comes into scope from a SOW query filter, the entire record must be sent. AMPS can not send a `delta_publish` message to update the record because there is no previous record to update on the client. Since Peyton Hillis was not a member of the New York Giants previously, the entire record, not the delta, had to be sent to the GUI client.

The `oof` message serves to notify a client that a record which previously met their SOW query filter criteria is no longer in the domain of the original query. In the previous example, at the time the `sow_and_delta_subscribe` command was issued, Brandon Jacobs was a running back for the New York Giants and was included as part of the original SOW query filter result set. When the trade was executed and Brandon Jacobs was traded to the Cleveland Browns, he no longer met the SOW query filter criteria (he was no longer on the roster for the New York Giants), thus signaling to AMPS that an `oof` message needed to be sent to update the client.

After all of the statistics have been updated and player deals have been accounted for in the above examples, the final client table looks like the diagram shown in [Figure 16.8](#)

In review, the key points for this example are to demonstrate the following points using a concrete example:



Figure 16.8: The final client table

1. OOF messages can be used as a powerful tool for ensuring that client data is updated in a consistent and timely manner.
2. The `sow_and_delta_subscribe` command works in a similar manner to the `sow_and_subscribe` command, but can offer substantial savings in network bandwidth by eliminating redundant data; and
3. Messages which didn't previously meet a SOW query filter criteria must be sent in full to the client.

Chapter 17

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 contains a command-line client, `spark`, which can be used to run queries, place subscriptions and publish data. While it can be used for each of these purposes, `spark` is provided as a useful tool for checking the status of the AMPS engine.

17.1 `amps_sow_dump`

`amps_sow_dump` is a utility used to inspect the contents of a SOW topic store.

Options and Parameters

Table 17.1: Parameters for `amps_sow_dump`.

Option	Description
<code>filename</code>	Filename of the SOW topic store.
<code>--version</code>	Show the version number of the program and exit.

Continued on next page

Table 17.1 -- continued from previous page

Option	Description
-h, --help	Show the help message and exit.
-n LIMIT	Maximum number of records to print per file.
-v, --verbose	Print record metadata for records and file summary.
-e, --escape	Escape special characters in record data and header.
-d DELIMITER	Prints only the record data using the provided ASCII character value as the record delimiter [default: 10 for newline].
--sizing-chart	Print memory sizing chart for efficiency comparison (experimental).

Usage



`amps_sow_dump` expects a filename at a minimum in order to complete the SOW topic store dump process.

The example below shows a simple sow dump with the `-e` flag set to make the header, message and field separators readable. Each key which exists in the `order.sow` file is dumped out to stdout. This output can easily be redirected to a new file, or piped into another program for further analysis.

```
%> ./amps_sow_dump -e ./order.sow

49=ABROKER\x0156=AFUNDMGR\x0152=2011-09-10T00:37:12\x01↵
  20=IBM\x0121=1000\x0122=93.17\x01
49=ABROKER\x0156=AFUNDMGR\x0152=2011-09-10T00:37:12\x01↵
  20=QCOM\x0121=100000\x0122=34.82\x01
49=ABROKER\x0156=AFUNDMGR\x0152=2011-09-10T00:37:12\x01↵
  20=AAPL\x0121=1000\x0122=97.73\x01
49=ABROKER\x0156=AFUNDMGR\x0152=2011-09-10T00:37:12\x01↵
  20=BRCM\x0121=100000\x0122=17.33\x01
49=ABROKER\x0156=AFUNDMGR\x0152=2011-09-10T00:37:12\x01↵
  20=HP\x0121=1000\x0122=22.32\x01
49=ABROKER\x0156=AFUNDMGR\x0152=2011-09-10T00:37:12\x01↵
  20=CSCO\x0121=1000\x0122=15.95\x01
49=ABROKER\x0156=AFUNDMGR\x0152=2011-09-10T00:37:12\x01↵
  20=DELL\x0121=100\x0122=9.26\x01
49=ABROKER\x0156=AFUNDMGR\x0152=2011-09-10T00:37:12\x01↵
  20=INTC\x0121=1000\x0122=13.98\x01
```

The next example shows the output from the `--sizing-chart` flag. This is feature can be useful in tuning AMPS memory usage and performance. The `Record Size` with the asterisk shows the current `Record Size` setting and allows an AMPS administrator to compare memory usage efficiency along with the potential for a multi-record penalty.



This feature is currently listed as experimental, so changing AMPS record size configuration based on the results may not necessarily help performance, and could hurt performance in some cases.

```
%> ./amps_sow_dump --sizing-chart ./order.sow
```

Record Size	Store	Efficiency	Multirecord
128	1024 B	100.00%	0
256	2.00 KB	50.00%	0
384	3.00 KB	33.33%	0
512*	4.00 KB	25.00%	0
640	5.00 KB	20.00%	0
768	6.00 KB	16.67%	0
896	7.00 KB	14.29%	0
1024	8.00 KB	12.50%	0
1152	9.00 KB	11.11%	0
1280	10.00 KB	10.00%	0
1408	11.00 KB	9.09%	0
1536	12.00 KB	8.33%	0
1664	13.00 KB	7.69%	0
1792	14.00 KB	7.14%	0
1920	15.00 KB	6.67%	0

17.2 amps_journal_dump

The AMPS journal dump utility is used in examining the contents of an AMPS journal file for debugging and program tuning.

Options and Parameters

Table 17.2: Parameters for `amps_journal_dump`.

Option	Description
<code>filename</code>	Filename of the AMPS journal file.
<code>--version</code>	Show the program version number and exit.
<code>-h, --help</code>	Show the program help message and quit.
<code>-l LIMIT</code>	limit range of output to entries N:M where N is the first entry and M is the last entry. Passing in a single value, M, will return the first M results.

17.3 `ampserr`

AMPS contains a utility to expand and examine error messages which may be observed in the logs. `ampserr` 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.

Options and Parameters

Table 17.3: Parameters for `amps_err`.

Option	Description
<code>error</code>	the error code to look up. This can also be a regular expression.

Usage

The following example shows the output of the "00-0001" error message.

```
%> ./ampserr 01-0001
AMPS Message 00-0001 [level = info]

  DESCRIPTION : AMPS Copyright message.

  ACTION      : No recommended action available.

Found 1 error matching '00-0001'.
```

The following example will return all message which begin with "00-".



For the sake of brevity, not all messages which match this query are printed in this manual.


```
%> ./ampserr 00-  
  
AMPS Message 00-0000 [level = trace]  
  
DESCRIPTION : Internal log message used by AMPS  
development team. If you see this message  
logged, please notify AMPS support.  
  
ACTION      : No recommended action available.  
  
AMPS Message 30-0000 [level = warning]  
  
DESCRIPTION : AMPS internal thread monitoring has  
detected a thread that hasn't made progress  
and appears 'stuck'. This can happen with long  
operations or a bug within AMPS.  
  
ACTION      : Monitor AMPS and if these 'stuck'  
messages continue, then a restart of the engine  
could be the only way to resolve it. If it  
appears busy (high CPU utilization) then it  
could be a long operation (large query filter.)
```

The following example will return all error messages.



For the sake of brevity, not all messages which match this query are printed in this manual.

```
%> ./ampserr .  
  
AMPS Message 00-0000 [level = trace]  
  
DESCRIPTION : Internal log message used by AMPS  
development team. If you see this message  
logged, please notify AMPS support.  
  
ACTION      : No recommended action available.  
  
AMPS Message 30-0000 [level = warning]  
  
DESCRIPTION : AMPS internal thread monitoring  
has detected a thread that hasn't made  
progress and appears 'stuck'. This can  
happen with long operations or a bug
```

```
within AMPS.
```

```
ACTION   : Monitor AMPS and if these 'stuck'  
           messages continue, then a restart of the  
           engine could be the only way to resolve it.  
           If it appears busy (high CPU utilization)  
           then it could be a long operation (large  
           query filter.)
```

17.4 spark

AMPS contains a command-line client, `spark`, which can be used to run queries, place subscriptions and publish data. While it can be used for each of these purposes, `spark` is provided as a useful tool for checking the status of the AMPS engine.

The `spark` utility and its accompanying source code is available in the `api/client/` directory of the AMPS install location. `spark` is currently written for `java`, `python` and `c#`. The source is provided as a best practices guide for implementing a simple AMPS client using the api provided.

To run `spark`, typing `spark` at the command line will display the help screen and given an idea of the features provided by sparks.

```
%> ./spark  
=====  
- Spark - AMPS client utility -  
=====  
Usage:  
  
  spark help [command]  
  
Supported Commands:  
  
  help  
  publish  
  sow  
  sow\_and\_subscribe  
  subscribe  
  
Example:  
  
%> ./spark help sow  
  
Returns the help and usage information for the  
  'sow' command.
```

`spark` requires that a supported command is passed as an argument. Within each supported command, there are additional unique requirements and options available to change the behavior of `spark` and how it interacts with the AMPS engine.

For example, if more information was needed to run a `publish` command in `spark`, the following would display the help screen for the `publish` feature in `spark`.

```
%>./spark help publish
=====
- Spark - AMPS client utility -
=====
Usage:

    spark publish [options]

Required Parameters:

    server -- AMPS server to connect to
    topic  -- topic to query
    type   -- message type to use (fix, xml)

Options:
    delimiter -- decimal value of separator character
                for messages. Default is 10 (LF)
    delta    -- use delta publish
    file     -- file to publish records from,
                standard in when omitted

Example:

%> ./spark publish -type fix -server localhost:9003
    -topic Trades -file data.fix

Connects to the AMPS instance listening on port 9003 ←
and publishes records
found in the 'data.fix' file to topic
'Trades'.
```

Chapter 18

Operation and Deployment

This chapter of the *Operations Guide* contains guidelines and best-practices to help plan and prepare an environment which will meet the demands that AMPS is expected to manage.

18.1 Capacity Planning

Sizing an AMPS deployment can be a complicated process that includes many factors including configuration parameters used for AMPS, the data used within the deployment, and how the deployment will be used. This section presents guidelines that you can use in sizing your host environment for an AMPS deployment given what needs to be considered along every dimension: Memory, Storage, CPU, and Network.

18.1.1 Memory

Beyond storing its own binary images in system memory, AMPS also tries to store its SOW and indexing state in memory to maximize the performance of record updates and SOW queries.

AMPS needs less than 1GB for its own binary image and initial start up state for most configurations. In the worst-case, because of indexing for queries, AMPS may need up to twice the size of messages stored in the SOW. And, finally AMPS needs some amount of memory reserved for the clients connected to it. While the per connection overhead is a tunable parameter based on the Slow Client Disconnect settings (see the best practices later in this chapter) it is advised to use 50MB per connected client.

This puts the worst-case memory consumption estimate at:

Example:

Table 18.1: Memory estimation equation.

S = Average SOW Message Size		1GB + (2S * M) + (C * 50MB)
M = Number of SOW Messages		
C = Number of Clients		

Table 18.2: Example memory estimation.

S = 1024		1GB + (2 * 1024 * 20,000,000) + (200 * 50MB)
M = 20,000,000		
C = 200		

≈ 52GB

An AMPS deployment expected to hold 20 million messages with an average message size of 1KB and 200 connected clients would consume 52GB. Therefore, this AMPS deployment would fit within a host containing 64GB with enough headroom for the OS under most configurations.

18.1.2 Storage

AMPS needs enough space to store its own binary images, configuration files, SOW persistence files, log files, transaction log journals, and slow client offline storage, if any. Not every deployment configures a SOW or transaction log, so the storage requirements are largely driven by the configuration.

Log files

Log file sizes vary depending on the log level and how the engine is used. For example, in the worst-case, `trace` logging, AMPS will need at least enough storage for every message published into AMPS and every message sent out of AMPS plus 20%.

For `info` level logging, a good estimate of AMPS log file sizes would be 2MB per 10 million messages published.

Logging space overhead can be capped by implementing a log rotation strategy which uses the same file name for each rotation. This strategy effectively truncates the file when it reaches the log rotation threshold to prevent it from growing larger.

SOW

When calculating the SOW storage, there are a couple of factors to keep in mind. The first is the average size of messages stored in the SOW, the number of messages stored in the SOW and the `RecordSize` defined in the configuration file. Using these values, it is possible to estimate the minimum and maximum storage requirements for the SOW:

Table 18.3: Minimum SOW size.

Min = Minimum SOW size	Min = S * M
S = Average SOW Message Size	
M = Number of SOW Messages	

Table 18.4: Maximum SOW size.

Max = Maximum SOW Size	Max = (S + R) * M
S = Average SOW Message Size	
R = Record Size	
M = Number of SOW Messages	

The storage requirements should be between the two values above, however it is still possible for the SOW to consume additional storage based on the unused capacity configured for each SOW topic.

For example, in an AMPS configuration file which has the `InitialSize` is set to 1000 messages and the `RecordSize` is set to 1024, the SOW for this topic will consume 1MB with no messages stored in the SOW. Pre-allocating SOW capacity in chunks is more efficient for the operating system, storage devices, and helps amortize the SOW extension costs over more messages.

It is also important to be aware of the maximum message size that AMPS can hold in the SOW. The maximum message size is calculated in the following manner:

Table 18.5: Maximum Message Size allowed in SOW.

Max = Maximum Message Size	Max = (R * I) - 40bytes
R = SOW Topic RecordSize	
I = SOW Topic IncrementSize	

This calculation says that the maximum message size that can be stored in the sow in a single message storage is the `RecordSize` multiplied by the `IncrementSize` minus 40 bytes for the record header information.

Other Storage Considerations

The previous sections discuss the scope of sizing the storage, however scenarios exist where the performance of the storage devices must also be taken into consideration.

One such scenario is the following use case in which the AMPS transaction log is expected to be heavily used. If performance greater than 1000 messages/second is required out of the AMPS transaction log, experience has demonstrated that flash storage (or better) would be recommended. Magnetic hard disks lack the performance to produce results greater than this.

18.1.3 CPU

SOW queries with content filtering make heavy use of CPU based operations, and as such, CPU performance directly impacts the content filtering performance and rates at which AMPS processes messages. The number of cores within a CPU largely determines how quickly SOW queries execute.

AMPS contains optimizations which are only enabled on recent 64-bit x86 CPU's. To achieve the highest level performance, consider deploying on a CPU which includes support for the SSE 4.2 instruction set.

To give an idea of AMPS performance, repeated testing has demonstrated that a moderate query filter with 5 predicates can be executed against 1KB messages at more than 1,000,000 messages per second, per core on an Intel i7 3GHz CPU. This applies to both subscription based content filtering and SOW queries. Actual messaging rates will vary based on matching ratios and network utilization.

18.1.4 Network

When capacity planning a network for AMPS, the requirements are largely dependent on the following factors:

- average message size
- the rate at which publishers will publish messages to AMPS
- the number of publishers and the number of subscribers.

AMPS requires sufficient network capacity to service inbound publishing as well as outbound messaging requirements. In most deployments, outbound messaging to subscribers and query clients has the highest bandwidth requirements due to the increased likeliness for a "one to many" relationship of a single published message matching subscriptions/queries for many clients.

Estimating network capacity requires knowledge about several factors, including but not limited to: the average message size published to the AMPS instance, the number of messages published per second, the average expected match ratio per subscription, the number of subscriptions, and the background query load. Once these key metrics are known, then the necessary network capacity can be calculated:

Table 18.6: Network capacity formula

R = Rate	$R * S(1 + M * S) + Q$
S = Average Message Size	
M = Match Ratio	
S = Number of Subscriptions	
Q = Query Load	

Where `QueryLoad` is:

Table 18.7: Network capacity formula

M_q = Messages Per Query		$M_q * S * Q_s$
S = Average Message Size		
Q_s = Queries Per Second		

Example:

In a deployment which is required to process published messages at a rate of 5000 messages per second, with each message having an average message size of 600 bytes. The expected match rate per subscription is 2% (or 0.02) with 100 subscriptions. The deployment is also expected to process 5 queries per minute (or $\frac{1}{12}$ queries per second), with each query expected to return 1000 messages.

$$5000 * 600B * (1 + 0.02 * 100) + (1000 * 600B * \frac{1}{12}) \approx 9MB/s \approx 72Mb/s$$

Based on these requirements, this deployment would need at least 72Mb/s of network capacity to achieve the desired goals. This analysis demonstrates AMPS by it self would fall into a 100Mb/s class network. It is important to note, this analysis does not examine any other network based activity which may exist on the host, and as such a larger capacity networking infrastructure than 100Mb/s would likely be required.

18.2 Linux Operating System Configuration

This section covers some settings which are specific to running AMPS on a Linux Operating System.

18.2.1 ulimit

The `ulimit` command is used by a Linux administrator to get and set user limits on various system resources.

ulimit -c It is common for an AMPS instance to be configured to consume gigabytes of memory for large SOW caches. If a failure were to occur in a large deployment it could take seconds (maybe even hours, depending on storage performance and process size!) to dump the core file. AMPS has a minidump reporting mechanism built in that collects information important to debugging an instance before exiting. This minidump is much faster than dumping a core file to disk. For this reason, it is recommended that the per user core file size limit is set to 0 to prevent a large process image from being dumped to storage.

ulimit -n The number of file descriptors allowed for a user running AMPS needs to be at least double the sum of counts for the following: connected clients, SOW topics and pre-allocated journal files.

Minimum: 1024

Recommended: 16834

18.2.2 /proc/sys/fs/aio-max-nr

Each AMPS instance requires AIO in the kernel to support at least 16384 plus 8192 for each SOW topic in simultaneous I/O operations. `aio-max-nr` setting is global to the host and impacts all applications, and as such this value needs to be set high enough to service all applications using AIO on the host.

Minimum: 65536

Recommended: 1048576

To view the value of this setting, as root you can enter the following command:

```
cat /proc/sys/fs/aio-max-nr
```

To edit this value, as root you can enter the following command:

```
sysctl -w fs.aio-max-nr=1048576
```

This command will update the value for `/proc/sys/fs/aio-max-nr` and allow 1,048,576 simultaneous I/O operations, but will only do so until the next time the machine is rebooted. To make a permanent change to this setting, as a root user, edit the `/etc/sysctl.conf` file and either edit or append the following setting:

```
fs.aio-max-nr = 1048576
```

18.2.3 /proc/sys/fs/file-max

Each AMPS instance needs file descriptors to service connections and maintain file handles for open files. This number needs to be at least double the sum of counts for the following: connected clients, SOW topics and pre-allocated journal files. This `file-max` setting is global to the host and impacts all applications, so this needs to be set high enough to service all applications on the host.

Minimum: 262144

Recommended: 6815744

To view the value of this setting, as root you can enter the following command:

```
cat /proc/sys/fs/file-max
```

To edit this value, as root you can enter the following command:

```
sysctl -w fs.file-max=6815744
```

This command will update the value for `/proc/sys/fs/file-max` and allow 6,815,744 concurrent files to be opened, but will only do so until the next time the machine is rebooted. To make a permanent change to this setting, as a root user, edit the `/etc/sysctl.conf` file and either edit or append the following setting:

```
fs.file-max = 6815744
```

18.3 Best Practices

This section covers a selection of best practices for deploying AMPS.

18.3.1 Monitoring

AMPS exposes the statistics available for monitoring via a RESTful interface, known as the [Monitoring Interface](#), which is configured as the administration port. This interface allows developers and administrators to easily inspect various aspects of AMPS performance and resource consumption using standard monitoring tools.

At times AMPS will emit log messages notifying that a thread has encountered a deadlock or stressful operation. These messages will repeat with the word "stuck" in them. AMPS will attempt to resolve these issues, however after 60 seconds of a single thread being stuck, AMPS will automatically emit a minidump to the previously configured minidump directory. This minidump can be used by 60East support to assist in troubleshooting the location of the stuck thread or the stressful process.

Another area to examine when monitoring AMPS is the `last_active` monitor for the processors. This can be found in the `/amps/instance/processors/all/last_active` url in the monitoring interface. If the `last_active` value continually increases for more than one minute and there is a noticeable decline in the quality of service, then it may be best to fail-over and restart the AMPS instance.

18.3.2 SOW Parameters

Choosing the ideal `InitialSize`, `IncrementSize`, and `RecordSize` for your SOW topic is a balance between the frequency of SOW expansion and storage space efficiency. A large `InitialSize` will preallocate space for records on start up, however this value may end up being too large, which would result in wasted space.

An `IncrementSize` that is too small results in frequent extensions of your SOW topic to occur. These frequent extensions can have a negative impact on the rate at which AMPS is able to process messages.

If the `IncrementSize` is large, then the risk of the SOW resize impacting performance is reduced, however this has a trade-off of reduced space utilization efficiency.

A good starting point for the `InitialSize` setting is 20% of the total messages a topic is expected to have, with `IncrementSize` being set to 10% of the total messages. This will minimize the number SOW size extensions while converging to a storage efficiency greater than 90%.

The `RecordSize` trade-offs are unique to the `InitialSize` and `IncrementSize` configuration discussed previously. A `RecordSize` that is too large results in space which will be wasted in each record. A `RecordSize` value which is too small and will result in AMPS using more CPU cycles managing space within the SOW.

If performance is critical and space utilization is a lesser concern, then consider using a `RecordSize` which is 2 standard deviations above your average message size. If storage space is a greater limiting factor, then look at the sizing histogram feature of the `amps_sow_dump` utility for guidance when sizing (see [Section 17.1](#) for more information).

18.3.3 Slow Clients

Clients which are unable to consume messages faster or equal to the rate messages are being sent to them are "slow clients". By default, AMPS queues messages for a slow client in memory to grant the slow client the opportunity to catch up. However, scenarios commonly arise where a client can be "over-subscribed" to the point it can not consume messages as fast as messages are being sent to it.

Within AMPS configuration there are several options to address and tune the performance of slow clients. The first tunable parameter for slow clients is the `ClientBufferThreshold`, which determines the number of bytes that can queue up for a client before AMPS will start queueing or "off-lining" the additional messages to disk. This setting should be sufficiently large to hold any query that a client could issue. Typically when AMPS prepares the messages for a client, it is common for query results to be queued in memory but immediately dequeued when the messages are sent to the client.



To prevent potential unbounded memory growth, by default `SlowClientDisconnect` and `ClientOffline` are enabled with `ClientBufferThreshold` set to 50MB and an offline file size limit of 1G.

For example, if a client is expected to have a maximum query size of 30,000 messages and the average message size is 1KB, then having `ClientBufferThreshold` set to 40MB would be a good initial configuration. This would cap the memory consumption per client at approximately 50MB

(assuming a standard 10MB of additional client-specific overhead such as filters and subscription information.)

Once AMPS exceeds the `ClientBufferThreshold` of queued messages for a client, AMPS will start enqueueing the messages to disk. AMPS writing the messages to disk has now changed a potential unbounded memory-growth problem into a potentially unbounded storage-growth problem. To deal with the potential problem of a client not responding while AMPS has started enqueueing its messages to disk, AMPS provides the `ClientOfflineThreshold` configuration parameter. This allows an AMPS administrator to tune the message threshold which AMPS will store on disk before disconnecting a slow client. For example, if we want to store at most 200MB of offlined data for a slow client, then we would set the `ClientOfflineThreshold` to 200MB.



When using AMPS within a development environment where a client consumer could be paused during debugging, it is often helpful to set the `Offlining` and `SlowClientDisconnect` thresholds larger than would normally exist in a production environment, or even turning the slow client disconnect feature off. This will reduce or prevent AMPS from disconnecting a client while it is in the process of testing or debugging a feature.

Listing 18.1 shows an example configuration of a `Transport` with `SlowClientDisconnect` enabled (`true`). In this example, a slow client will first start to offline messages when it falls behind by 10,000 messages as determined by the `ClientOfflineThreshold` setting. The client may continue to fall further behind, however, when the offline message queue reaches 4MB in size the client will be disconnected. This is determined by the `ClientBufferThreshold` limit which is set to 4194304.

```

1 <Transports>
2 <Transport>
3   <Name>fix-tcp</Name>
4   <Type>tcp</Type>
5   <InetAddr>10200</InetAddr>
6   <ReuseAddr>true</ReuseAddr>
7   <MessageType>nvfix</MessageType>
8   <ClientBufferThreshold>4194304</ClientBufferThreshold>
9   <ClientOffline>enabled</ClientOffline>
10  <ClientOfflineThreshold>10000</ClientOfflineThreshold>
11  <SlowClientDisconnect>true</SlowClientDisconnect>
12 </Transport>
13 </Transports>

```

Listing 18.1: Example of FIX Transport with Slow Client Configuration

When monitoring for slow clients, a good place to look is in the `queue_depth_out` or in the `queue_bytes_out` parameters which are

tracked for each client in the monitoring interface. These parameters are located in `/amps/instance/clients/*/queue_depth_out` or in `/amps/instance/clients/*/queued_bytes_out` where the `*` is the client being monitored. If the `queued_bytes_out` or `queue_depth_out` values continually increase and a client notices that they have fallen behind, then that client should be disconnected. Additionally, if this happens repeatedly, then investigate proper usage of the `SlowClientDisconnect` functionality within AMPS using the guidelines listed previously, or examine the selectivity of the filters to improve consumption and performance of the client.

18.3.4 Minidump

AMPS includes the ability to generate a minidump file which can be used in support scenarios to attempt to troubleshoot a problematic instance. The minidump captures information prior to AMPS exiting and can be obtained much faster than a standard core dump (see [Section 18.2.1](#) for more configuration options). By default the minidump is configured to write to `/tmp`, but this can be changed in the AMPS configuration by modifying the `MiniDumpDirectory`.

Generation of a minidump file occurs in the following ways:

1. When AMPS detects a crash internally, a minidump file will automatically be generated.
2. When a user clicks on the `minidump` link in the `amps/instance/administrator` link from the administrator console (see [Section D.2](#) for more information).
3. By sending the running AMPS process the `SIGQUIT` signal.
4. If AMPS observes a single stuck thread for 60 seconds, a minidump will automatically be generated. This should be sent to AMPS support for evaluation along with a description of the operations taking place at the time.

Chapter 19

Monitoring Interface

AMPS includes a monitoring interface which is useful for examining many important aspects about an AMPS instance. This includes health and monitoring information for the AMPS engine as well as the host AMPS is running on. All of this information is designed to be easily accessible to make gathering performance and availability information from AMPS easy.

For a reference regarding the fields and their data types available in the AMPS monitoring interface, see [Appendix D](#).

19.1 Configuration

The AMPS monitoring interface is defined in the configuration file used on AMPS start up. Below is an example configuration of the `Admin` tag.

```
1 <!-- Configure the admin/stats HTTP server -->
2 <Admin>
3   <FileName>stats.db</FileName>
4   <InetAddr>localhost:8085</InetAddr>
5   <Interval>10s</Interval>
6 </Admin>
```

In this example `localhost` is the hostname and `8085` is the port assigned to the monitoring interface. This chapter will assume that

```
http://localhost:8085/
```

is configured as the monitoring interface URL.

The `Interval` tag is used to set the update interval for the AMPS monitoring interface. In this example, statistics will be updated every 10 seconds.



It is important to note that by default AMPS will store the monitoring interface database information in system memory. If the AMPS instance is going to be up for a long time, or the monitoring interface statistics interval will be updated frequently, it is strongly recommended that the `FileName` setting be specified to allow persistence of the data to a local file. See [Section C.3.2](#) for more information.

The administrative console is accessible through a web browser, but also follows a Representational State Transfer (RESTful) URI style for programmatic traversal of the directory structure of the monitoring interface.

The root of the AMPS monitoring interface URI contains two child resources - the `host` URI and the `instance` URI - each of which is discussed in greater detail below. The `host` URI exposes information about the current operating system devices, while the `instance` URI contains statistics about a specific AMPS deployment.

19.2 Time Range Selection

AMPS keeps a history of the monitoring interface statistics, and allows that data to be queried. By selecting a leaf node of the monitoring interface resources, a time based query can be constructed to view a historical report of the information. For example, if an administrator wanted to see the number of messages per second consumed by all processors from midnight UTC on October 12, 2011 until 23:25:00 UTC on October 10, 2011, then pointing a browser to

```
http://localhost:8085/amps/instance/processors/all/  
messages_received_per_sec?t0=20111129T0&t1=20111129T232500
```

will generate the report and output it in the following plain text format (note: entire dataset is not presented, but is truncated).

```
20111130T033400,0  
20111130T033410,0  
20111130T033420,0  
20111130T033430,94244  
20111130T033440.000992,304661  
20111130T033450.000992,301078  
20111130T033500,302755  
20111130T033510,308922  
20111130T033520.000992,306177  
20111130T033530.000992,302140  
20111130T033540.000992,302390  
20111130T033550,307637  
20111130T033600.000992,310109  
20111130T033610,309888
```

```
20111130T033620,299993
20111130T033630,310002
20111130T033640.000992,300612
20111130T033650,299387
```



All times used for the report generation and presentation are ISO-8601 formatted. ISO-8601 formatting is of the following form: `YYYYMMDDThhmmss`, where `YYYY` is the year, `MM` is the month, `DD` is the day, `T` is a separator between the date and time, `hh` is the hours, `mm` is the minutes and `ss` is the seconds. Decimals are permitted after the `ss` units.



The date-time range can be used with the plain text (html), comma-separated (csv) and XML formats - which are discussed below.

19.3 Output Formatting

The AMPS monitoring interface offers several possible output formats to ease the consumption of monitoring reporting data. The possible options are XML, CSV and RNC output formats - each of which is discussed in more detail below.

19.3.1 XML Document Output

All monitoring interface resources can have the current node, along with all child nodes list its output as an XML document by appending the `.xml` file extension to the end of the resource name. For example if an administrator would like to have an XML document of all of the currently running processors - including all the relevant statistics about those processors - then the following URI will generate that information:

```
http://localhost:8085/amps/instance/processors/all.xml.
```

The document that is returned will be similar to the following:

```
1 <amps>
2   <instance>
3     <processors>
4       <processor id="all">
5         <bytes_published>776198025</bytes_published>
```



```

6      <bytes_published_per_sec>2561190</↵
      bytes_published_per_sec>
7      <client_publishes>4372952</client_publishes>
8      <client_publishes_per_sec>14430</↵
      client_publishes_per_sec>
9      <description>AMPS Aggregate Processor Stats</↵
      description>
10     <last_active>271542</last_active>
11     <matches_found>4372952</matches_found>
12     <matches_found_per_sec>14430</↵
      matches_found_per_sec>
13     <messages_received>69900009</messages_received>
14     <messages_received_per_sec>229998</↵
      messages_received_per_sec>
15   </processor>
16 </processors>
17 </instance>
18 </amps>

```

Appending the `.xml` file extension to any AMPS monitoring interface resource will generate the corresponding XML document.

19.3.2 CSV Document Output

Similar to the XML document output discussed above, the `.csv` file extension can be appended to any of the leaf node resources to have a CSV file generated to examine those values. This can also be coupled with the time range selection to generate reports. See [Section 19.2](#) for more details on time range selection.

Below is a sample of the `.csv` output from the monitoring interface from the following URL:

```

http://localhost:8085/amps/instance/processors/all/
messages_received_per_sec.csv?t0=20111129T0

```

This resource will create a file with the following contents:

```

20111130T033400,0
20111130T033410,0
20111130T033420,0
20111130T033430,94244
20111130T033440.000992,304661
20111130T033450.000992,301078
20111130T033500,302755
20111130T033510,308922
20111130T033520.000992,306177
20111130T033530.000992,302140
20111130T033540.000992,302390
20111130T033550,307637
20111130T033600.000992,310109

```

```
20111130T033610,309888
20111130T033620,299993
20111130T033630,310002
20111130T033640.000992,300612
20111130T033650,299387
20111130T033700.000992,304548
```

19.3.3 RNC Document Output

AMPS supports generation of an XML schema via the Relax NG Compact (RNC) specification language. To generate an RNC file, enter the following URL in a browser <http://localhost:port/amps.rnc> and AMPS will display the RNC schema.

To convert the RNC schema into an XML schema, first save the RNC output to a file:

```
%> wget http://localhost:9090/amps.rnc
```

The output can then be converted to an xml schema using `trang` (available at <http://code.google.com/p/jing-trang/>) with

```
trang -I rnc -O xsd amps.rnc amps.xsd
```

Chapter 20

High Availability

High Availability in AMPS is provided through a primary site replication mechanism. The "High Availability" chapter discusses how replication works within AMPS, demonstrates how messages flow in a high availability configuration, and then describes a few possible scenarios where AMPS replication can be deployed to minimize downtime and increase the reliability of AMPS. AMPS replication can also be used to push data to AMPS instances in other regions to improve the response times of downstream clients in higher latency environments.

20.1 Transaction Log

AMPS has a transaction log that can be used to store messages for later replay or for replication between AMPS instances. The transaction log configuration can contain topics and content filters for messages that should be persisted to the transaction log. Each message stored to the transaction log is given a unique bookmark that specifies its position within the transaction log.

```
<TransactionLog>
  <JournalDirectory>./journal</JournalDirectory>
  <MinJournalSize>10MB</MinJournalSize>
</TransactionLog>
```

When the transaction log is enabled, clients can request a replay of messages in the transaction log by issuing a `subscribe` command and providing a `Bookmark`. The replay will start on the message after the bookmark, or at the first stored transaction if the epoch bookmark "0" is used.



An epoch bookmark can be used as the `bookmark` during the `logon` process which will replay the transaction log from the beginning. Use of the epoch is accomplished by using the `bookmark` number 0 (zero) when issuing the `logon` command.



While there are similarities between a bookmark subscription used for replay and a SOW query, the transaction log and SOW are independent features that can be used separately.

20.2 Replication

Messages stored to a transaction log can be replicated to downstream AMPS instances. AMPS supports two forms of replication links: synchronous and asynchronous which control when publishers of messages are sent `persisted` acknowledgments.

AMPS won't return a `persisted` acknowledgment to the publisher for a message until the message has been stored to the transaction log, SOW, and all downstream synchronous replication destinations. See [Figure 20.1](#) and [Figure 20.2](#) for a comparison between `synchronous` and `asynchronous` replication and what it means to the delivery of `persisted` acknowledgments back to the publisher.

In `synchronous` replication, as in figure [Figure 20.1](#), the `persisted` acknowledgment isn't returned to the publisher until after the downstream secondary AMPS instance has acknowledged that it persisted the message.

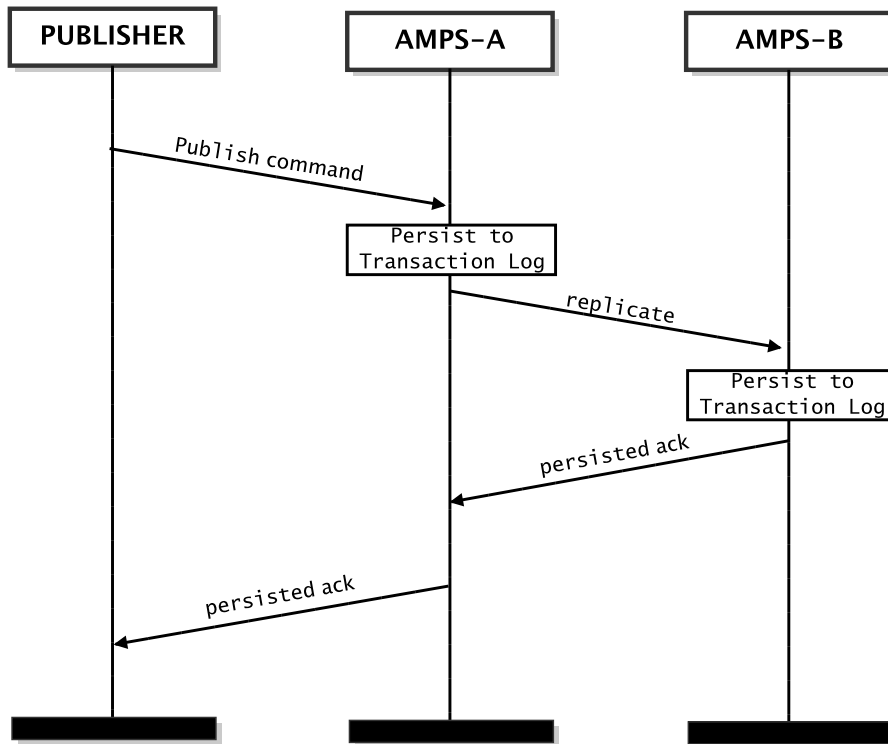


Figure 20.1: Synchronous Persistence Acknowledgment

In asynchronous replication, the primary is free to send the *persisted acknowledgment* back to the publisher as soon as the message is safely persisted to the transaction log and SOW, when configured, as in figure [Figure 20.2](#).

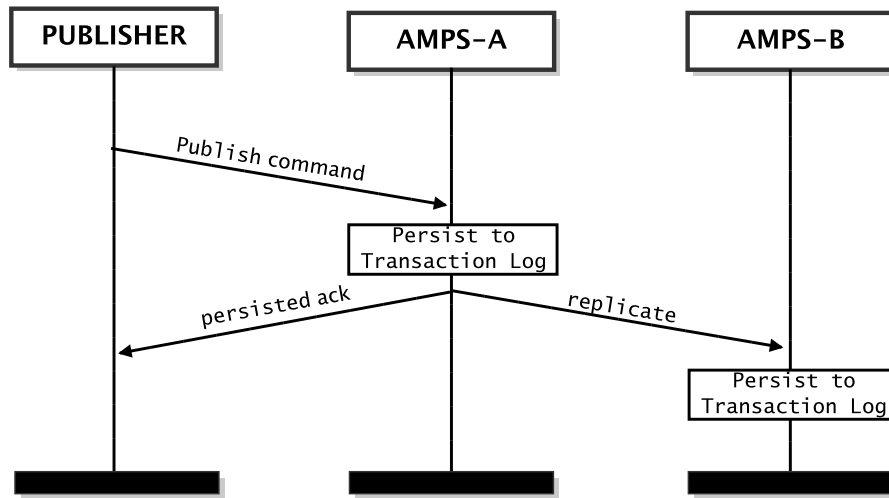


Figure 20.2: Asynchronous Persistence Acknowledgment

20.3 Bookmarks

Bookmarks are unique identifiers that map to individual transactions in the transaction log. Downstream subscribers will see a `bookmark` field attached as part of the message header of returned messages matching their subscriptions. The `bookmark` will only be included in messages that have been stored to the transaction log. This allows a subscriber to disconnect from AMPS, reconnect at a later time and then subscribe again passing in `bookmark`. This will inform the AMPS engine that the client would like to resume from point of the last message that matched its subscription.

20.4 Publishing for High Availability

Publishing in a high availability environment in AMPS has a few unique qualities when compared to publishing in a non-replicated configuration. Initially when a publisher issues a `logon` command to AMPS, the returned `ack` message will contain a `SeqNo`. The `SeqNo` is used to inform the publisher of the monotonically increasing sequence number AMPS last saw while messages were published. The publisher then uses that `SeqNo` to increment and publish additional messages into AMPS.

As the publisher is publishing messages, AMPS will return `Ack` messages back to the publisher, however there is not a one-to-one relationship of messages published to `ack` messages returned. AMPS will withhold acknowledging these

messages immediately, and will instead send a single `ack` message which corresponds to the `SeqNo` of the most recent persisted message. This single `ack` serves to imply that AMPS has persisted and acknowledged all messages with a `SeqNo` which is less than the `SeqNo` of the acknowledged message. This network conserving process of minimizing `ack` messages is known as *conflation*. This process is illustrated in Figure 20.3.

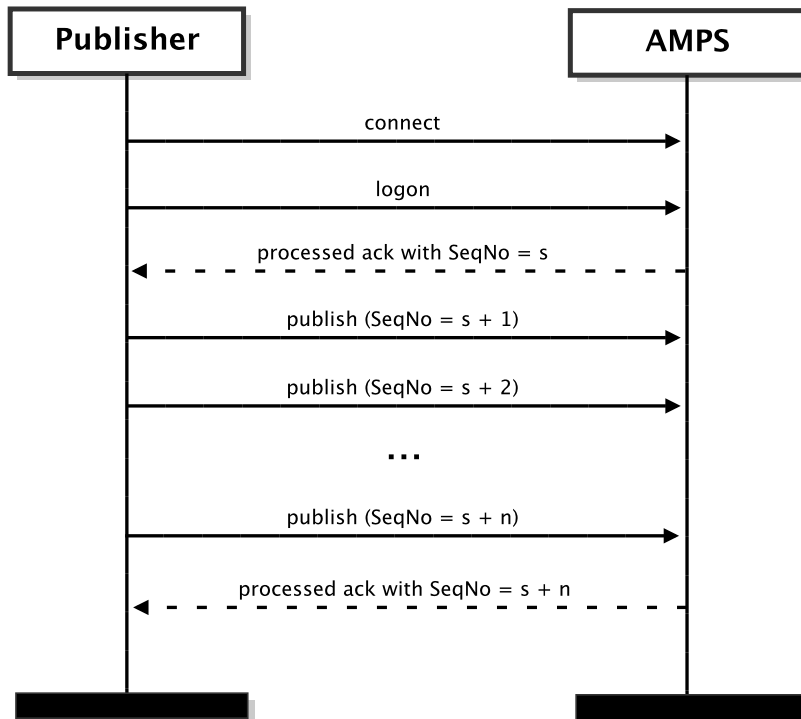


Figure 20.3: Persisted Ack strategy.



While a publisher is waiting for a conflated `ack` message to be returned, the publisher is responsible for saving all messages which have been sent to AMPS, but have not yet received an `ack` message.

As described in the [Replication](#) section, when a publisher receives an `ack` with a status of `persisted`, this means the message has been persisted to the local `log`, transaction log at a minimum (`async` replication) or it has been persisted to all downstream replication destinations (`sync` replication). When a message has been persisted to all replication clients, it is referred to as "fully persisted."

In some cases, publishing messages with a `SeqNo` that AMPS has seen before can result in an `ack` message confirming that the message was a `duplicate`. This message implies that the message has already been fully persisted, and AMPS has not taken any action on this message with regard to updating the message being persisted.

20.5 Subscribing for High Availability

When subscribing in a High Availability environment where AMPS has a Transaction Log configured, it is the responsibility of the subscribing client to keep track of the `BkMrk` of all incoming messages. A bookmark is used when a client is forced to reconnect, logon and subscribe again. If the client has the bookmark from the last message it received AMPS will replay the transaction log from the message associated with the bookmark up to the most recent message persisted in the transaction log. After the transaction log has been replayed, the subscriber will be rejoined with the subscription message flow.

Once a client has subscribed and caught up with the messages which are being replayed from the transaction log, AMPS provides two unique ways of rejoining a client to the subscription message flow. The default behavior for a subscription in a high availability environment - known as a "tail", since it acts like the UNIX "tail" command on the transaction log - is for AMPS to send messages to a client once those messages have been persisted to the transaction log.



The advantage of the 'tail' approach is that a subscribing client will never miss a message delivery, but there is an increased latency in that the client will have to wait for matching subscribed messages to commit to the transaction log.

An alternate subscription type is a `live` subscription where AMPS will send the messages to the subscribing client before they are persisted in the transaction log. To configure this, a subscription command is created with an `Options` field set to `live`. A `live` subscription has the benefit of reducing the latency of a message between the publisher sending the message and the client receiving the message, however there is risk associated with this type of subscription. This risk is discussed below.

20.5.1 Potential for Risk During a Live Subscription

As already noted, the transaction log stores the ordering of the messages, so under normal, non-fail-over cases subscriptions will always see the same messages in the same order.

The hazards of the `live` subscription come into play during a fail-over. They can have re-ordering issues as well as duplicate and/or missing messages for

any transaction that was sent over the `live` subscription to your client ahead of it being durably persisted to the AMPS transaction log.

For this scenario we will assume there are two `live` bookmark subscriptions on a single client: `Sub1` and `Sub2`.

There are also two publishers: one that publishes messages `A1..A1000`, and another which publishes messages `B1..B1000`. These publishers will be called `Publisher A` and `Publisher B`, respectively. Once the two publishers have completed, the result is that 2000 messages were published in total. While unlikely in reality, we are going to assume the publishers are perfectly timed so that AMPS processes the transactions and writes them to the topics in alternating order: `A1, B1, A2, B2, ..., A1000, B1000`.

AMPS sends all 2000 messages to `Sub1` and `Sub2`. However, at some point AMPS is brought down while attempting to write message `A501` to the transaction log, it follows that AMPS fails to write all remaining messages to the transaction log. AMPS is then restarted and has messages `A1` through `B500` safely in its transaction log. At this point any one of the three following scenarios are possible.

Scenario 1 - Subscribers reconnect first

The subscribers `Sub1` and `Sub2` re-subscribe with the bookmark that came in on `B1000`. AMPS does not know about this bookmark yet, so it places the subscriptions at the last transaction in the transaction log - message `B500` and makes `Sub1` and `Sub2` a `live` subscription.

Next, the publishers re-connect and re-publish the records missing from AMPS `A501..B1000`. This causes the subscribers to receive `A501.. B1000` again. To further complicate things, the duplicated messages could be in a different order given the timing between the publishers.

Scenario2 - Publisher A reconnects first

`Publisher A` connects, logs on and is sent a `SeqNo` of `A500`, which causes `Publisher A` to re-publish the missing messages into AMPS `A501..A1000`.

The subscribers `Sub1` and `Sub2` re-subscribe with the bookmark that came in on `B1000`. AMPS does not know about this bookmark yet, so it places the subscriptions at the last transaction in the transaction log - `A1000` - and makes `Sub1` and `Sub2` a `live` subscription.

`Publisher B` re-connects, logs on and is sent a `SeqNo` of `B500`, which causes `Publisher B` to re-publish the missing messages into AMPS `B501..B1000`. The subscribers receive `B501..B1000` again, however it is important to note they do not receive `A501..A1000` again.

Scenario3 - Publisher B reconnects first

`Publisher B` connects, logs on and is sent a `SeqNo` of `A500`, which causes `Publisher B` to re-publish the missing messages into `amps B501..B1000`.

The subscribers `Sub1` and `Sub2` re-subscribe with the bookmark that came in on `B1000`. AMPS recognizes this bookmark, so it places the subscriptions at that point in the transaction log. `Publisher A` re-connects, logs on and receives a `SeqNo` of `A500` which prompts it to re-publish messages `A501..A1000`. This causes `Sub1` and `Sub2` to receive messages `A501..A1000` again.

20.5.2 Alternate Scenario Demonstrating Live Subscription

For this scenario, we will assume a single publisher named `Pub`, which is publishing alternating messages `A1, B1, A2, B2, ..., A1000, B1000`. Each of these messages are being published to different topics. In this scenario it is important to point out that within AMPS, the ordering is guaranteed by the store and forward mechanism on the publisher.

Next, a single client connects, logs on and creates two `live` bookmark subscriptions, each on a separate topic - `A` and `B`. At some point during this subscription, AMPS is shutdown and restarted.

Once AMPS restarts and the subscribing client attempts to reconnect, logon and re-subscribe with the bookmark of the last message it received, it is possible for the subscriber's bookmark could be ahead of AMPS transaction log. In this case, AMPS has not seen the bookmark requested by the subscriber, so the subscriber will be immediately joined with the live stream of messages. This operation is safe since the publisher is guaranteeing the same order of messages.



If there is the possibility of the publisher(s) reconnecting and re-publishing messages in an order that is different than what was originally sent before AMPS was restarted, then it is recommended that a subscribing client implements one of the `BookmarkStore` interfaces. The `BookmarkStore` interface is covered in each of the language Developer Guides, available with the AMPS installation documentation.

20.6 Deployment Examples

There are a varied number of use cases and deployment possibilities for AMPS, so the three most common deployments will be covered here. Before discussing

some of the variations in deployments, it's important to discuss what all of these unique instances have in common.

In all deployments of AMPS, a message is considered persisted if and only if it has been committed to the transaction log, SOW and all down stream synchronous replication destinations.

Persisted acknowledgments have the ability to be conflated (also called "message conflation"). In this context conflation means that a single message can be used to describe more than one message. For example, if a publisher has published 1,000 messages, sequentially numbered 1 through 1,000, and the publisher expects a *persisted* acknowledgment back from each message published, AMPS may only send an acknowledgment for message 1,000. This conflated acknowledgment serves to notify the publisher that AMPS has received and persisted all messages up to and including message 1,000. AMPS can conflate the persisted acknowledgments, because it knows that the messages coming from the publisher must be monotonically increasing.

20.6.1 Single AMPS Deployment

Much of this User Guide so far has focused on the single AMPS deployment with one or more publishers and one or more clients subscribed to the instance (Figure 20.4 is an illustration of this configuration). While AMPS attempts to isolate clients and publishers from any sort of failure, in this configuration if the AMPS instance should fail, the only possible recovery is to wait for the instance to recover. No other alternative is provided for the publisher or the clients.

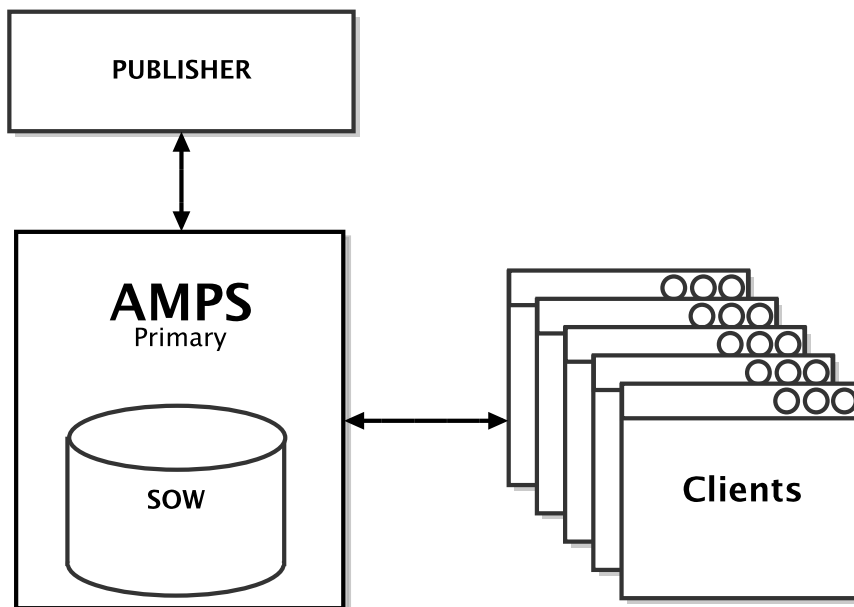


Figure 20.4: Single AMPS instance

20.6.2 Simple High Availability Pair

In a simple high availability configuration of AMPS, the two instance will appear as a single instance to any publishers publishing messages or clients subscribing to messages. One distinction between the single AMPS instance described previously is that when a `persisted` acknowledgment is returned from a publish message, it means the message has been persisted in the AMPS primary instance and in the AMPS secondary instance.

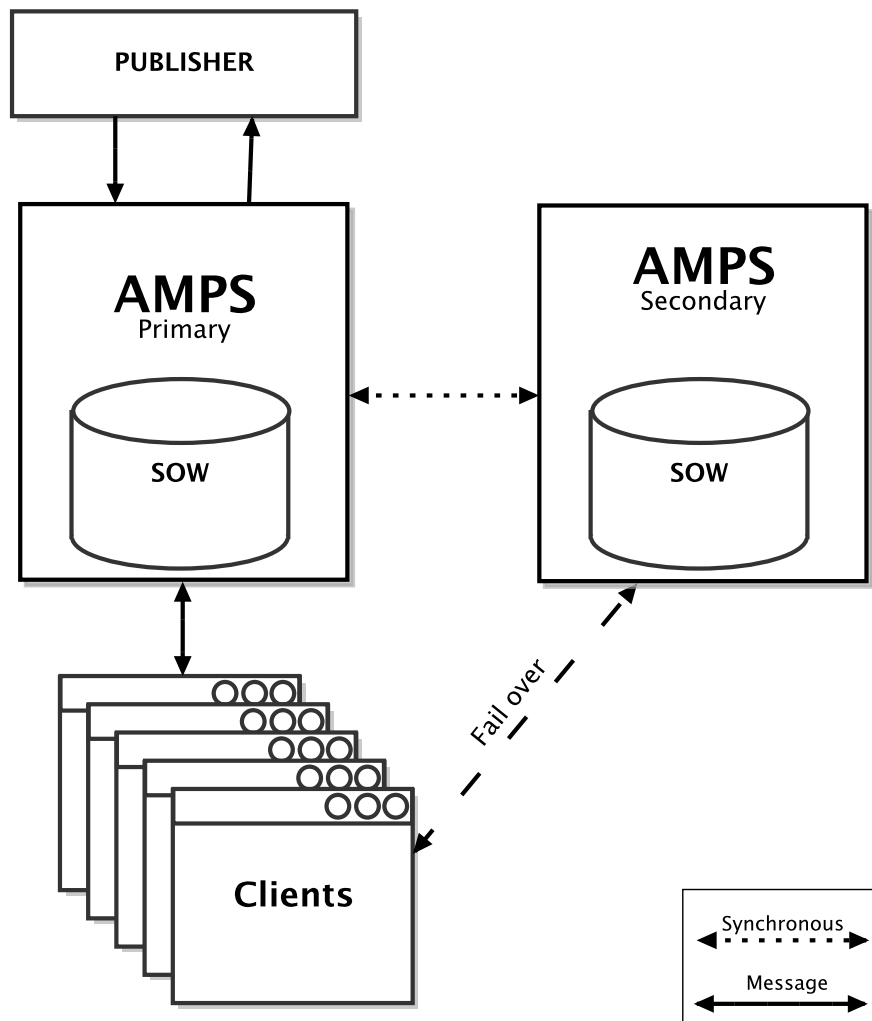


Figure 20.5: Simple High Availability Pair

Figure 20.5 shows an example of this configuration. In this scenario, when the publisher publishes a message to the AMPS primary instance, the message is synchronously sent to the AMPS secondary instance (as illustrated by the dotted line). The AMPS secondary instance returns a `persisted` acknowledgment back to the AMPS primary instance. The `persisted` acknowledgment is then

returned back to the publisher - thus ensuring that the original message has been persisted in all downstream SOW topics and/or transaction logs. This follows the message acknowledgment process described in [Figure 20.1](#) for a synchronous configuration or [Figure 20.2](#) for an asynchronous configuration.

The clients subscribed in [Figure 20.5](#) show a two way message connection between themselves and the primary instance of AMPS. Should the primary instance of AMPS fail, the clients will take the fail-over path to the AMPS secondary instance. Since the AMPS secondary instance has been kept synchronized with the AMPS primary instance, the clients are able to continue processing messages with no data loss.

Now that the concept of a replicated AMPS installation has been introduced, it is important to take a look back at some of the architectural requirements needed to make all the various communications work. This section will start by examining the requirements placed on an AMPS publisher to ensure the best possible experience with AMPS.

20.6.3 Publisher Responsibilities and Guarantees

Publisher will issue a `logon` command upon successfully connecting to an AMPS instance. The `logon` command will contain the `ClientName`, a name unique to all connected AMPS replicas that can be used to identify this publisher. The `logon` command will request a return of a `processed` acknowledgment message. For example, a publisher would issue the following FIX command during a `logon`:

```
Command=logon;ClientName=Publisher01;AckType=processed;
```

Once AMPS receives the `logon` request, a `processed` acknowledgment message will be returned. This response message will contain the `SeqNo` of the last record persisted. If no messages have been persisted for the unique client identifier (`ClientName`), then the `SeqNo` will be 0 (zero).

A publisher with no messages persisted by the AMPS instance will need to start publishing with the `SeqNo` set to 1. The sequence number transmitted with each `publish` is to be incremented by 1 for each message published. If a publisher is in an environment where it is unable recreate the published messages, then a `persisted` acknowledgment should be requested with each `publish`. The publisher should then retain messages until a `persisted` acknowledgment message is returned from the AMPS instance which notifies the publisher of the last persisted message. Messages will be persisted in order, so if a publisher receives a `persisted` acknowledgment with `SeqNo` 1000, then this can be interpreted as the AMPS instance notifying the publisher that all messages up to the message with `SeqNo` 1000 have been persisted. This process using a single message to imply the acknowledgment of all prior unacknowledged messages is known as conflation.

A publisher which receives a non-zero `SeqNo` from an AMPS instance means messages have been persisted. In this scenario, publishers are expected to discard messages up to and including the `SeqNo` returned and begin publishing

messages which occurred after the discarded messages. It is possible that AMPS will return a `persisted` acknowledgment which is considerably lower than the `SeqNo` observed in a previously connected `logon` session. This would happen if the connection between the publisher and AMPS instance was lost before AMPS could return a `persisted` acknowledgment, yet AMPS could continue persisting the records even without the connection. Messages published with a `SeqNo` lower than that reported by AMPS on the `logon` ack will be discarded with an error logged to the AMPS logs. Other than a small performance penalty for sending and discarding the messages, this scenario is idempotent.

20.6.4 Complex High Availability with Regional Replication

An example of a complex high availability deployment is one the instances of AMPS are geographically distant from each other. In this scenario, AMPS is being used to replicate messages from multiple publishers into a single instance of AMPS (AMPS-A). AMPS-A is then responsible for replication to two alternate sites, one which is reasonably close (AMPS-B) and another AMPS instance which is over a high latency network connection (AMPS-C). AMPS-C then replicates messages to an instance which has a high latency network (AMPS-D). AMPS-B is also assigned to replicate incoming messages to AMPS-C to guarantee message delivery from the publishers to AMPS-C.

AMPS-A and AMPS-B have a synchronous replication defined between the two. This means that any publishers publishing messages to AMPS-A will not receive `persisted` acknowledgments until AMPS-A has received a `persisted` acknowledgment from AMPS-B. AMPS-A will not wait for AMPS-C to return a `persisted` acknowledgment because the replication definition between those two instances is asynchronous. This means that AMPS-A will expect a `persisted` acknowledgment from AMPS-C, but AMPS-A does not wait for the `persisted` acknowledgment from AMPS-C before returning the `persisted` acknowledgment to the message publisher.

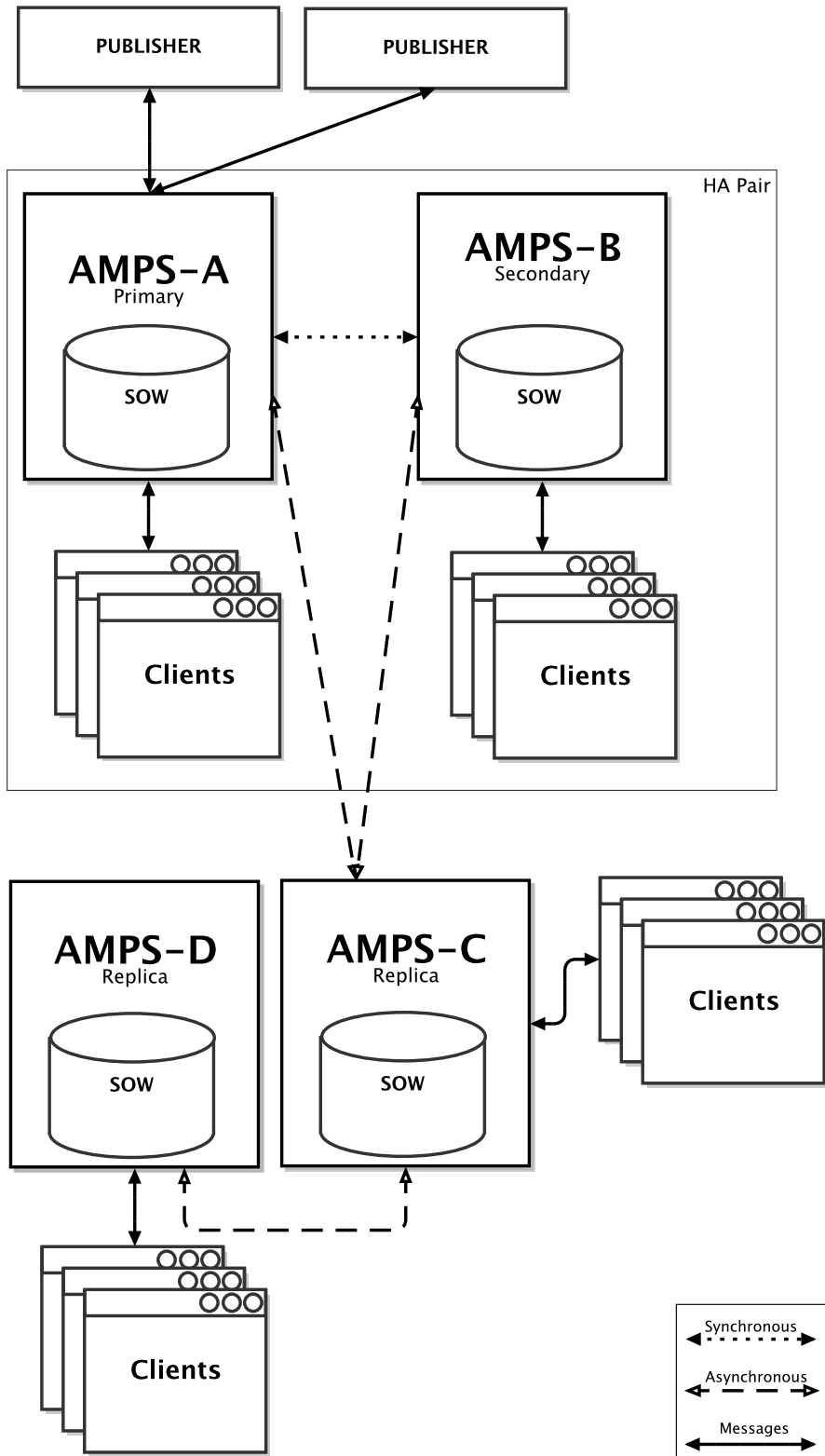


Figure 20.6: Complex High Availability with Regional Replication

20.7 Potential Points of Failure

In the example shown in [Figure 20.6](#), there are several points of failure which will be listed here along with possible corrective actions and overall risks.

20.7.1 Publisher Loses Connection

Corrective Action(s) Reconnecting the publisher(s) which lost connection corrects this problem. Publishers will re-transmit messages which were not acknowledged by AMPS.

Risk Messages which are re-transmitted have an associated cost to them, but once this process has been completed AMPS and the publisher should be able to resume normal operation. The worst case scenario is one in which the publisher has lost all record of which messages have been acknowledged and which have not. If a publisher re-transmits all messages, the downstream AMPS instances will not re-transmit messages which were previously acknowledged. This behavior can protect limited network resources from a high volume of redundant published messages.

20.7.2 Primary (AMPS-A) Instance Becomes Unavailable

Corrective Action(s) Restarting the primary instance resolves this problem in many cases. The primary instance will read its transaction log to determine which messages have been acknowledged and which have not. The AMPS instance will then resume re-transmission of all messages which have not been acknowledged.

If restarting the primary AMPS instance does not resolve the issue, the publisher(s) should be reconnected to the secondary instance of AMPS in the replica. Publishing can resume by re-publishing all messages which have not been acknowledged as `processed`. Clients should also be migrated off of the primary AMPS instance and reconnected to the secondary AMPS instance.

Risks The recovery time for the restarted instance, while variable, depends on the amount of time necessary for the primary AMPS instance to read its transaction log to recover its state. The larger the number of messages which have been processed by AMPS, the longer the potential recovery time. This recovery time can also cause a backlog of messages from the publishers to accumulate while they are waiting for the AMPS instance(s) to resume processing message.

If the primary AMPS instance is down for an extended period and the reconnection to the secondary instance of AMPS takes an extended period of time, then any publishers could experience upstream queuing of messages, causing a propagation of slow message processing.

If during the recovery the AMPS transaction log is corrupted, moved or missing, the AMPS instance will consider this a catastrophic failure and will fail to start.

20.7.3 Secondary (AMPS-B) Instance becomes Unavailable

Corrective Action(s) Restarting the secondary AMPS instance should be performed similar to the process described in the previous section. If the secondary AMPS instance cannot be restarted, then clients should reconnect to the primary AMPS instance and resume subscriptions. Within the primary AMPS instance, the replication link to the secondary AMPS instance should be downgraded to `asynchronous` in the monitoring console. This will allow the primary AMPS instance to send `persisted` acknowledgments back to the publisher. If the secondary AMPS instance is restarted and is able to resume message processing, then replication will restart between the primary and secondary instances, however the secondary AMPS instance will not prevent `persisted` acknowledgments from being sent to the publisher.

Risks The risks to this scenario are the same as the previous section where the primary AMPS instance has become unavailable.

20.7.4 AMPS-C Instance becomes Unavailable

Corrective Action(s) Restart the AMPS-C instance - this will behave in a similar manner as the primary AMPS (AMPS-A) instance restart in the earlier section.

Risks While the AMPS-C instance is down, clients in the second region will not have a local instance to connect to. Additionally the AMPS-D instance will not be receiving messages from the AMPS-C instance.

20.7.5 AMPS-D Instance becomes Unavailable

Corrective Action(s) Restart the AMPS-D instance - this will behave in a similar manner as the primary AMPS (AMPS-A) instance restart in the earlier section.

Risks Clients in the alternate region will not have a local instance to connect to while the AMPS-D instance is down.

20.8 Configuration

Configuring AMPS for a replication environment requires defining some specific replication configuration information on both ends of an AMPS replication. This

section will cover how to configure both a *replication source* and a *replication destination*, each of which has unique requirements.

20.8.1 Replication Source

The AMPS replication source instance is configured using a `Replication` section with one or more `Destination` sections. Each replication destination is defined with a unique `Name`, the type of replication `SyncType`, one or more `Topic` sections which will be replicated from the source to the destination, and the destination transport `Transport`.

All AMPS replication instances use a proprietary transport when defining the `Type` in the configuration, the `amps-transport` transport type, to send messages between replication instances. The `amps-transport` type is required to be defined in the `Transport` section.

```

1 <AMPSConfig>
2
3 ...
4
5 <TransactionLog>
6   <JournalDirectory>./amps-A/journal</JournalDirectory>
7   <PreallocatedJournalFiles>1</PreallocatedJournalFiles
8     >
9   <MinJournalSize>10MB</MinJournalSize>
10  <BatchSize>128</BatchSize>
11  <Topic>
12    <Name>orders</Name>
13    <MessageType>fix</MessageType>
14  </Topic>
15  <FlushInterval>100ms</FlushInterval>
16 </TransactionLog>
17
18 <Replication>
19   <Destination>
20     <Name>B</Name>
21     <Transport>
22       <InetAddr>localhost:10005</InetAddr>
23       <Type>amps-replication</Type>
24     </Transport>
25     <Topic>
26       <Name>orders</Name>
27       <MessageType>fix</MessageType>
28     </Topic>
29     <SyncType>sync</SyncType>
30   </Destination>
31 </Replication>
32 ...

```

```

33
34 </AMPSConfig>

```

Listing 20.1: Replication Source Example

20.8.2 Replication Destination

An AMPS instance that receives replicated messages must define a `Transport` with which to receive replicated messages.

On the *replication destination* instance, messages are delivered over a `sync` replication, then a `TransactionLog` must be defined. A replication destination will maintain the last message it has received and upon recovery will inform the replication source where to begin replication. It is possible that some message might be duplicated because the recording of the last received message may be delayed from what has actually been delivered from the replication source and published to subscribers.



It is important to note that when implementing an `async` replication instance without a `TransactionLog`, re-sync with an upstream instance can potentially lead to delivery of duplicated messages.

Configuring a `TransactionLog` with replication destinations requires that topic filtering criteria be inclusive of all messages that might be replicated. For example, if Topic A is replicated, then Topic A must be specified in the `TransactionLog` section. It is perfectly fine to replicate a subset of messages that are stored in the transaction log but this does not work in reverse, i.e. replicated messages cannot be sent unless they are written to the transaction log.

```

1 ...
2
3 <Transports>
4   <Transport>
5     <Name>amps-replication</Name>
6     <Type>amps-replication</Type>
7     <InetAddr>localhost:10005</InetAddr>
8     <ReuseAddr>true</ReuseAddr>
9   </Transport>
10 </Transports>
11
12 <TransactionLog>
13   <JournalDirectory>./amps-B/journal/B/</
    JournalDirectory>
14   <MinJournalSize>100MB</MinJournalSize>

```

```
15 <BatchSize>128</BatchSize>
16 <Topic>
17   <Name>topic</Name>
18   <MessageType>fix</MessageType>
19 </Topic>
20 </TransactionLog>
21
22 ...
```

Listing 20.2: Replication Destination Example

Chapter 21

Sample Use Cases

To further your understanding of AMPS, we provide some sample use cases that highlight how multiple AMPS features can be leveraged in larger messaging solutions. For example, AMPS is often used as a back-end persistent data store for client desktop applications.

The provided use case shows how a client application can use the AMPS command `sow_and_subscribe` to populate an order table that is continually kept up-to-date. To limit redundant data from being sent to the GUI, we show how you can use a delta subscription command. You will also see how to improve performance and protect the GUI from over-subscription by using the `TopN` query limiter along with a `stats` acknowledgement.

21.1 View Server Use Case

Many AMPS deployments are used as the back-end persistent store for desktop GUI applications. Many of the features covered in previous chapters are unique to AMPS and make it well suited for this task.

In this example AMPS will be act as a data store for an application with the following requirements:

- allow users to query current order-state (SOW query)
- continually keep the returned data up to date by applying incremental changes (subscribe)

For purposes of highlighting the functionality unique to AMPS, we'll skip most of the details and challenges of GUI development.

21.1.1 Setup

For this example, let's configure AMPS to persist FIX messages to the topic `ORDERS`. We use a separate application to acquire the FIX messages from the market (or other data source) and publish them into AMPS. AMPS accumulates all of the orders in its SOW persistence, making the data available for the GUI clients to consume.

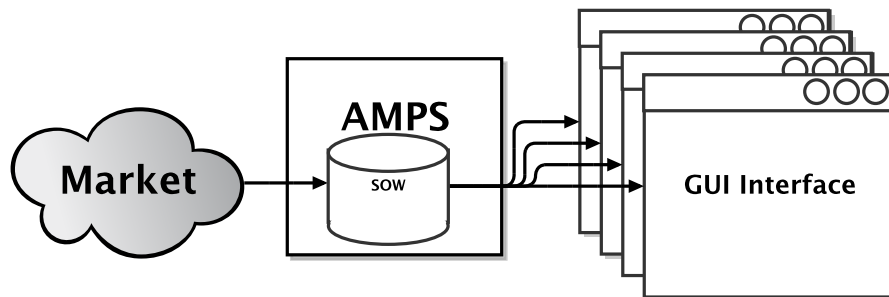


Figure 21.1: AMPS View Server Deployment Configuration

21.1.2 SOW Query and Subscription

The GUI will enable a user to enter a query and submit it to AMPS. If the query filter is valid, then the GUI displays the results in a table (aka "grid") and continually applies changes as they are published from AMPS to the GUI. For example, if the user wants to display active orders for `Client-A`, then they may use a query similar to this:

```
/11 = 'Client-A' AND /39 IN (0, 'A')
```

This filter matches all orders for `Client-A` that have FIX tag 39 (the FIX order status field) as 0 ('New') or 'A' ('Pending New').

From a GUI client, we want to first issue a query to pull back all current orders and, at the same time, place a subscription to get future updates and new orders. AMPS provides the `sow_and_subscribe` command for this purpose.



A more realistic scenario may involve a GUI Client with multiple tables, each subscribing with a different AMPS filter, and all of these subscriptions being managed in a single GUI Client. A single connection to AMPS can be used to service many active subscriptions if the subscription identifiers are chosen such that they can be demultiplexed during consumption.

The GUI issues the `sow_and_subscribe` command, specifying a topic of `ORDERS` and possibly other filter criteria to further narrow down the query results. Once the `sow_and_subscribe` command has been received by AMPS, the query returns to the GUI all messages in the SOW that, at the moment, match the topic and content filter. Simultaneously, a `subscription` is placed to guarantee that any messages not included in the initial query result will be sent after the query result.

The GUI client then receives a `group_begin` message from AMPS, signaling the beginning of a set of records returned as a result of the query. Upon receiving the initial SOW query result, this GUI inserts the returned records into the table, as shown in [Figure 21.2](#). Every record in the query will have assigned to it a unique `SowKey` that can be used for future updates.

The receipt of the `group_end` message serves as a notification to the GUI that AMPS has reached the end of the initial query results and going forward all messages from the subscription will be live updates.

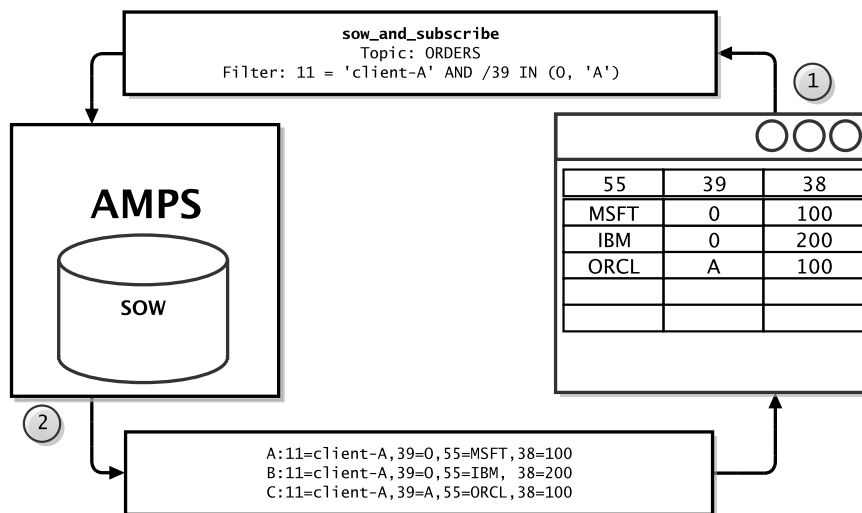


Figure 21.2: AMPS GUI Instance With `sow_and_subscribe`

Once the initial SOW query has completed, each `publish` message received by the GUI will be either a *new* record or an *update* to an existing record. The `SowKey` sent as part of each `publish` message is used to determine if the newly published record is an update or a new record. If the `SowKey` matches an existing record in the GUI's order table, then it is considered an update and should replace the existing value. Otherwise, the record is considered to be a new record and can be inserted directly into the order table.

For example, assume there is an update to order `C` that changes the order status (tag 39) of the client's `ORCL` order from `'A'` to `0`. This is shown below in [Figure 21.3](#).

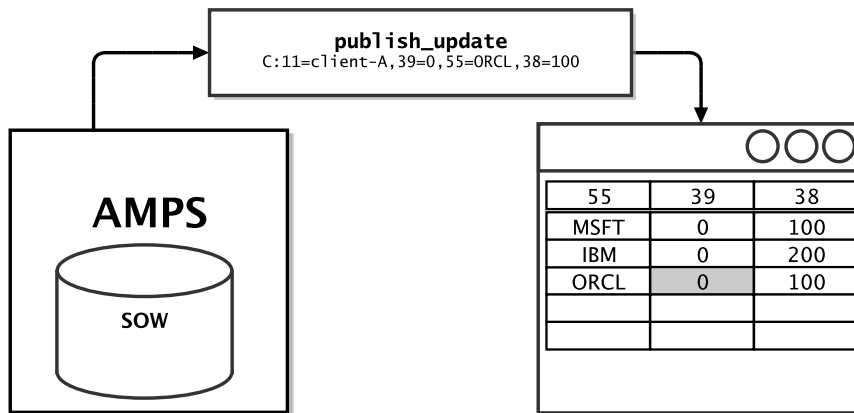


Figure 21.3: AMPS Message Publish Update

21.1.3 OOF Processing

Let's take another look at the original filter used to subscribe to the `ORDERS` `SOW` topic. A unique case exists if an update occurs in which an `ORDER` record status gets changed to a value other than 0 or 'A'. One of the key features of AMPS is `oof` processing, which ensures that client data is continually kept up-to-date. OOF processing is the AMPS method of notifying a client that a new message has caused a `SOW` record's state to change, thus informing the client that a message which previously matched their filter criteria no longer matches or was deleted. For more information about `oof` processing, see the [Out of Focus Message Processing \(OOF\)](#) chapter.

When such a scenario occurs, AMPS won't send the update over a normal subscription. If OOF processing is enabled within AMPS by specifying the `SendOOF` header option for this subscription, then updates will occur when previously matching records no longer match due to an update, expiration, or deletion.

For example, let's say the order for `MSFT` has been filled in the market and the update comes into AMPS. AMPS won't send the published message to the GUI because the order no longer matches the subscription filter; AMPS instead sends it as part of an `oof` message. This happens because AMPS knows that the previous matching record was sent to the GUI client prior to the update. Once an `oof` message is received, the GUI can remove the corresponding order from the orders table to ensure that users see only the up-to-date state of the orders which match their filter.

21.1.4 Conclusion and Next Steps

In summary, we've shown how a GUI application can use the `sow_and_subscribe` command to populate an order table, which is then continually kept up-to-date. AMPS can create further enhancements, such as those

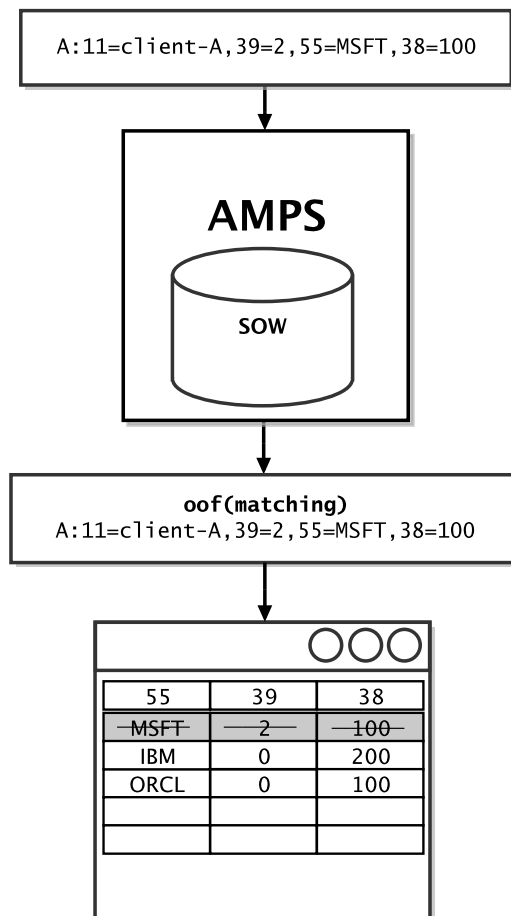


Figure 21.4: AMPS OOF Processing

described below, that improve performance and add greater value to a GUI client implementation.

sow_and_delta_subscribe

The first improvement that we can make is to limit redundant data being sent to the GUI, by placing a `sow_and_delta_subscribe` command instead of a `sow_and_subscribe` command. The `sow_and_delta_subscribe` command, which works with the FIX message type, can greatly reduce network congestion as well as decrease parsing time on the GUI client, yielding a more responsive end-user experience.

With a delta subscription, AMPS [Figure 21.3](#) sends to the GUI only the values that have changed: `C:39=0` instead of all of the fields that were already sent to the GUI during the initial SOW query result. This may seem to make little

difference in a single GUI deployment; but it can make a significant difference in an AMPS deployment with hundreds of connected GUI clients that may be running on a congested network or WAN.

TopN and Stats

We can also improve client-side data utilization and performance by using a `TopN` query limiter with a `stats` acknowledgment, which protects the GUI from over-subscription.

For example, we may want to put a 10,000 record limit on the initial query response, given that users rarely want to view the real-time order state for such a large set. If a `TopN` value of `10000` and an `AckType` of `stats` is used when placing the initial `sow_and_subscribe` command, then the GUI client would expect to receive up to 10,000 records in the query result, followed by a `stats` acknowledgment.

The `stats` is useful for tracking how many records matched and how many were sent. The GUI client can leverage the `stats` acknowledgment metrics to provide a helpful error to the user. For example, in a scenario where a query matched 130,000 messages, the GUI client can notify the user that they may want to refine their content filter to be more selective.

Appendix **A**

Header Field Reference

A.1 FIX Message Header - Sorted by Value

FIX Header Field	Name
20000	Command
20001	CommandId
20002	ClientName
20003	UserId
20004	TransmissionTime
20005	Topic
20006	Filter
20007	MessageType
20008	AckType
20009	SubscriptionId
20012	Expiration
20013	SendSubscriptionIDs
20014	DataOnly
20015	Heartbeat
20016	TimeoutInterval
20017	GracePeriod
20018	Status
20019	QueryID
20020	SendOutOfFocus
20021	LogLevel
20022	UseNamespaces
20023	BatchSize
20025	TopNRecordsReturned

FIX Header Field	Name
20029	SendEmpty
20031	MaximumMessages
20032	SowKeys
20033	SendKeys
20036	Sequence
20037	Bookmark
20038	Password
20054	RecordsDeleted
20055	RecordsReturned
20056	TopicMatches
20057	Matches
20058	MessageLength
20059	SowKey
20060	GroupSequenceNumber
20061	SubscriptionIds
20062	Reason
20063	MessageID

Table A.1: FIX Header Fields - sorted by FIX Value

A.2 FIX Message Header - Sorted by Name

FIX Header Field	Name
20008	AckTyp
20037	BkMrk
20023	BtchSz
20002	ClntName
20000	Cmd
20001	CmdId
20014	DatOnly
20012	Expn
20006	Filtr
20017	GrcPrd
20060	GrpSqNum
20015	Hrtbt
20021	LogLvl
20057	Matches
20063	MsgId
20058	MsgLen

FIX Header Field	Name
20007	MsgTyp
20031	MxMsgs
20038	PW
20019	Qld
20062	Reason
20054	RecordsDeleted
20055	RecordsReturned
20036	Seq
20029	SndEmpty
20033	SndKeys
20020	SndOOF
20013	SndSublds
20059	SowKey
20032	SowKeys
20018	Status
20009	Subld
20061	Sublds
20016	TmIntvl
20025	TopN
20056	TopicMatches
20005	Tpc
20004	TxmTm
20022	UseNS
20003	Usrld

Table A.2: FIX Header Fields - sorted by Name

A.3 XML Message Header - Sorted by Name

XML Header Field	Name
AckTyp	AckType
BkMrk	Bookmark
BtchSz	BatchSize
ClntName	ClientName
Cmd	Command
Cmdld	Commandld
DatOnly	DataOnly
Expn	Expiration
Fitr	Filter

XML Header Field	Name
GrcPrd	GracePeriod
GrpSqNum	GroupSequenceNumber
Hrtbt	Heartbeat
LogLvl	LogLevel
Matches	Matches
MsgId	MessageID
MsgLen	MessageLength
MsgTyp	MessageType
MxMsgs	MaximumMessages
PW	Password
QId	QueryID
Reason	Reason
RecordsDeleted	RecordsDeleted
RecordsReturned	RecordsReturned
Seq	Sequence
SndEmpty	SendEmpty
SndKeys	SendKeys
SndOOF	SendOutOfFocus
SndSubIds	SendSubscriptionIDs
SowKey	SowKey
SowKeys	SowKeys
Status	Status
SubId	SubscriptionId
SubIds	SubscriptionIds
TmIntvl	TimeoutInterval
TopN	TopNRecordsReturned
TopicMatches	TopicMatches
Tpc	Topic
TxmTm	TransmissionTime
UseNS	UseNamespaces
UsrId	UserId

Table A.3: XML Header Fields - sorted by Name

A.4 Header Fields - Sorted by Name

Name	Definition
AckType	Acknowledgement type for the given command. Type: any combination of: received, processed, completed, stats

Name	Definition
BatchSize	Specifies the number of messages that are batched together when returning a query result. Type: integer (default is 1)
Bookmark	A client originated identifier used to mark a location in journaled messages. See chapter Topic Replicas for more information. Type: string
ClientName	Used to identify a client. Useful for publishers that wish to identify the source of a publish, client status messages and for client heartbeats. Can be set with <code>logon</code> command. Type: string
Command	Command to be executed. Type: one of: <code>publish</code> , <code>delta_publish</code> , <code>delta_subscribe</code> , <code>subscribe</code> , <code>sow</code> , <code>sow_and_delta_subscribe</code> , <code>sow_and_subscribe</code> , <code>sow_delete</code> , <code>unsubscribe</code> , <code>heartbeat</code> , <code>start_timer</code> , <code>stop_timer</code> , <code>logon</code> ,
CommandId	Client specified command id. The <code>CmdId</code> is returned by the engine in response to client commands. Type: string
DataOnly	If <code>true</code> , only send raw data to subscriber for a matching publish message, i.e. do not include SOAP envelope. Type: boolean (<code>true</code> or <code>false</code>)
Expiration	Publish message expiration time if used in <code>publish</code> . Type: integer (seconds)
Filter	Content filter expression Type: string, should wrap in CDATA
GracePeriod	Grace period after heartbeat interval is exceeded before client is considered in error state. Type: integer (milli seconds)
GroupSequenceNumber	Group Sequence Number returned with each batch message of an SOW response. Type: integer
Heartbeat	Heartbeat command Type: one of: <code>start</code> , <code>stop</code> , <code>beat</code>
LogLevel	Set the log level Type: one of: <code>info</code> , <code>none</code>
Matches	Returned in the ack to an SOW query indicating number of matches Type: integer
MaximumMessages	Specifies the maximum number of messages within a batch publish. Type: integer greater than zero

Name	Definition
MessageID	Set by AMPS engine to tag every incoming message. Type: string, eg: MAMPS-XYZ
MessageLength	Sent with FIX formatted message data to indicate the number of bytes used by the message body. Type: integer
MessageType	Message transport type Type: string, one of <code>xml</code> , <code>fix</code> , <code>nvfix</code>
Password	Password used for authenticating with an AMPS server. Type: string
QueryID	SOW Query identifier set by client to identify a query. Type: string
Reason	Reason indicates textual failure message when ack message returns a Status of failure. Type: string
RecordsDeleted	Used in conjunction with the <code>stats ack</code> , this is the number of records deleted from the SOW when issuing a <code>sow_delete</code> command. Type: integer
RecordsReturned	Returned in the ack to an SOW query indicating number of records in the store. Type: integer
SendEmpty	If <code>true</code> , empty messages that are published will be forwarded to matching subscriptions. Type: boolean (<code>true</code> or <code>false</code>) default is <code>true</code>
SendKeys	Option to instruct AMPS that a client would like to receive the SowKey(s) back. Type: boolean
SendOutOfFocus	If <code>true</code> , Out-of-Focus messages are sent for the SOW query. Type: boolean (<code>true</code> or <code>false</code>)
SendSubscriptionIDs	If <code>false</code> , subscription identifiers will not be sent for a matched message. Type: boolean (<code>true</code> or <code>false</code>)
Sequence	An integer, 1 or greater, corresponding to the publish message sequence number. For more information see Chapter 13 on replication. Type: integer
SowKey	An SowKey will accompany each message returned in an SOW batch, for XML it will be contained in the "Msg" section. A SowKey may also be added to messages coming in on a subscription when the published message matches a record in the SOW. Type: unsigned long

Name	Definition
SowKeys	Comma separated list of List of SOW Key integer integers. Type: integers
Status	Used to indicate client status when client is being monitored for heartbeats. Type: one of: <code>stopped</code> , <code>alive</code> , <code>timed out</code> , <code>error</code>
SubscriptionId	The subscription identifier set by server when processing a subscription. Type: string, eg: <code>SAMPS-XYZ</code>
SubscriptionIds	Comma separated list of SubIds sent from AMPS engine to identify which client subscriptions match a given publish message. Type: string
TimeoutInterval	Used in conjunction with the heartbeat interval to set the timeout interval for a publisher. Type: integer
TopNRecordsReturned	The number of records to return. Note: If TopN is not equally divisible by the BtchSz, then more records will be returned so that the total number of records is equally divisible by the BtchSz setting. Type: unsigned integer
Topic	Topic Type: string
TopicMatches	Returned in the ack to an SOW query indicating number of topic matches. Type: integer
TransmissionTime	Transmission timestamp set by client. Type: ISO-8601 datetime
UseNamespaces	Use SOAP XML namespaces in all messages from the AMPS engine. Type: boolean (<code>true</code> or <code>false</code>)
UserId	Used to identify the user id of a command. Type: string

Table A.4: Header Fields - sorted by Name

Appendix **B**

Command Reference

This appendix includes a listing of all AMPS commands as well as the required and optional parameters.

B.1 `delta_publish`

Description

The `delta_publish` command is a way of publishing an incremental update to a record. If a client uses `delta_publish` to publish an update, AMPS first extracts the key fields from the record and does a look up for the record in the SOW. AMPS will then apply the update to the SOW record overwriting any field that has a newer value in the update and appending to the record any new fields that were not previously in the SOW record.

If `delta_publish` is used on a record that does not currently exist in the SOW or if it is used on a topic that does not have a SOW-topic store defined, then `delta_publish` will behave like a standard `publish` command.

A `delta_publish` is transparent to other clients and the merged record will be forwarded to matching subscriptions.

Header Fields

Table B.1 contains the header fields available to a `delta_publish` command.

Table B.1: Header fields used in a `delta_publish`

Field	Description
Command	Command to be executed. Value: <code>delta_publish</code>

Continued on next page

TableB.1 -- continued from previous page

Field	Description
Topic	Topic to place a subscription with.
AckType	Acknowledgment type for the given command. Value is a comma separated list of one or more of the following: none , received , processed , completed or stats
CommandId	If specified with an AMPS command requesting an ack, all requested ack messages will contain the CommandId in the ack response header.
Expiration	An <i>interval</i> used to define the lifetime of a delta_publish message.
Sequence	A monotonically increasing number used to identify published messages in a high availability environment.
TransmissionTime	An ISO-8601 datetime used to note the time the message is sent by client.

Returns

For a delta_publish message, AMPS will send acknowledgment messages for the following AckType fields: received , processed and persisted along with a populated Status header field describing the acknowledgment.

Table B.2 contains the AckType messages which can be returned by a delta_publish .

Table B.2: Ack types supported by delta_publish

ackType	Description
none	No ack message is returned. This is the default behavior.
received	The delta_publish message has been received.
persisted	When an AMPS engine returns an ackType of persisted this guarantees that 1) all downstream synchronous replication(s) all have acknowledged that the message(s) have been deleted from their respective SOW Topic(s). 2) when the delta_publish message has been sent to all available asynchronous replications.
processed	AMPS has processed the message(es) to be published to the SOW.
completed	Not supported at this time.
stats	Not supported at this time.

Errors

Any errors that occur during this command will be returned in the status of a `processed` acknowledgment and logged to the log file. Regardless of success or failure, the `processed` acknowledgment will only be returned if requested by including `processed` in the `AckType` field.

Examples

Below are some examples of the `delta_publish` command.

`delta_publish` command in FIX and NVFIX

Here is an example of FIX `delta_publish` messages that use ";" as the field separator and "|" for the header separator. For this example, the SOW topic "topic" has a key defined on /1 (all records sharing the same value for field "1" are defined to be the same record).

First, the following FIX record is published to AMPS using `delta_publish` :
"1=a;2=b;3=c;".

```
20000=delta_publish;20005=topic;|1=a;2=b;3=c;
```

Second, the following FIX record is published to AMPS using `delta_publish` :
"1=a;2=d;4=e;".

```
20000=delta_publish;20005=topic;|1=a;2=d;4=e;
```

After the 2nd `delta_publish` the SOW will contain a single record where "1=a" and will have the body of "1=a;2=d;3=c;4=e;". The "1=a" is common to both records and remains unchanged in the merged record. Field "2" changes values from "b" to "d" on the update. Field "3" remains set to value "c" and is retained in the merged record. Finally, field "4" is appended with value "e" to the end of the record leaving the merged result "1=a;2=d;3=c;4=e;".



FIX and NVFIX messages are the only message type supported by `delta_publish` .

B.2 delta_subscribe

Description

The `delta_subscribe` command is like the `subscribe` command except that subscriptions placed through `delta_subscribe` will only receive messages which have changed between the SOW record and the new update.

If `delta_subscribe` is used on a record which does not currently exist in the SOW or if it is used on a topic which does not have a SOW-topic store defined, then `delta_subscribe` behaves like a `subscribe` command.

Header Fields

Table B.3 contains the header fields available to a `delta_subscribe` command.

Table B.3: Header fields supported by `delta_subscribe`.

Field	Description
Command	Command to be executed. Value: <code>delta_publish</code>
Topic	Topic to place a subscription with.
AckType	Acknowledgment type for the given command. Value is a comma separated list of one or more of the following: <code>none</code> , <code>received</code> , <code>processed</code> , <code>completed</code> or <code>stats</code>
CommandId	If specified with an AMPS command requesting an <code>ack</code> , all requested <code>ack</code> messages will contain the <code>CommandId</code> in the <code>ack</code> response header.
DataOnly	A Boolean (<code>true</code> or <code>false</code>) to determine the type of data sent to the subscriber. A value of <code>true</code> will, for example, not include SOAP envelope.
Filter	CDATA wrapped string which is used as a content filter expression.
SendEmpty	Boolean (<code>true</code> or <code>false</code>) value used to determine whether empty messages which are published will be forwarded to matching subscriptions. The default value is <code>true</code> .
SendSubscriptionIds	Boolean (<code>true</code> or <code>false</code>) subscription identifiers will not be sent for a matched message.
TransmissionTime	An ISO-8601 datetime used to note the time the message is sent by client.

Returns

For a `delta_subscribe` message, AMPS will send acknowledgment messages for the following `AckType` fields: `received`, `processed`, `persisted` and `stats` along with a populated `Status` header field describing the acknowledgment.

Table B.4 contains the `AckType` messages which can be returned by a `delta_subscribe`.

Table B.4: Ack types supported by `delta_subscribe`

<code>ackType</code>	Description
<code>none</code>	No ack message is returned. This is the default behavior.
<code>received</code>	The <code>delta_subscribe</code> message has been received.
<code>persisted</code>	Not supported at this time.
<code>processed</code>	AMPS has compiled the filters for the <code>delta_subscription</code> message(es).
<code>completed</code>	The query has been completed. All messages arriving after the receipt of the <code>completed</code> acknowledgment message will be from published messages, and not from the SOW query.
<code>stats</code>	Returns an ack message with <code>Matches</code> , <code>TopicMatches</code> and <code>RecordsReturned</code> .

Errors

Any errors that occur during this command will be returned in the status of a `processed` acknowledgment and logged to the log file. Regardless of success or failure, the `processed` acknowledgment will only be returned if requested by including `processed` in the `AckType` field of the `delta_subscribe` message header.

Examples

Below are some examples of the `delta_subscribe` command. The following examples use ";" as the field separator and "|" as the header separator.

This first example is a FIX `delta_subscribe` command message

```
20000=delta_subscribe;20005=order;20008=processed;
```

This command will place a subscription for all publications on the `order` topic. For this example, the `order` topic has a SOW message key defined to be

/1.After the subscription is placed, another client publishes a new record
1=a;2=b;3=c;:

```
20000=delta_publish;20005=order;|1=a;2=b;3=c;
```

Since the message is new and is not currently in the SOW, the full message is sent to the subscriber: 1=a;2=b;3=c;. Finally, the publishing client publishes the following update to the record: 1=a;2=c;4=d;:

```
20000=delta_publish;20005=order;|1=a;2=c;4=d;
```

When the SOW record is updated, the subscriber receives only the delta 2=c;4=d; because only the 2 and 4 fields differed from what was previously in the SOW for the same record. The SOW record contains the entire record 1=a;2=c;3=c;4=d; because `delta_publish` was used. If `publish` had been used by the publishing client instead, the same result would be sent to the subscriber, but the SOW would be left to contain the last published record 1=a;2=c;4=d; instead of the merged record.

Notes

Because two messages are only deemed the same if their message keys are equivalent, `delta_subscribe` will always remove the message keys. For cases where it is necessary to know which record is which on the client side, use the `SowKey` field on the matching record to disambiguate the records.



The `delta_subscribe` command is only supported when using the FIX and NVFIX message type.

B.3 logon

Description

To help identify clients, it is recommended that clients send a `logon` command to the AMPS engine and specify a client name. This client name is used to identify a client and does not have to be unique as the AMPS engine does not use it for anything other than placing into the publish message for a client status event.

Header Fields

Table B.5 contains the header fields available to a `logon` command.

Table B.5: Header fields supported by `logon`.

Field	Description
Command	The <code>logon</code> command.
ClientName	A <code>string</code> identifier used to give a client a unique id.
AckType	Acknowledgment type for the given command. Value is a comma separated list of one or more of the following: <code>none</code> , <code>received</code> or <code>processed</code> .
Bookmark	The sequence ID of the last message received by the client. Passing in a <code>bookmark</code> will cause AMPS to replay the transaction log from the <code>bookmark</code> up to the most recent message persisted in the transaction log.
UserId	The username passed into the AMPS authentication and entitlement module.
Password	The password passed into the AMPS authentication and entitlement module.

Returns

A `logon` message which has specified an `AckType` of `received` or `processed` will receive an `ack` message to acknowledge the message receipt. If a client requests an acknowledgment message, the header will also contain the `ClientName` which was part of the original `logon` message.

When requested, the `logon` will return a `processed` `ack` which is used in determining if a client was successfully authenticated against a server which has an authentication module enabled. [Section E.3 - AMPS Guarantees](#) has more information on the returns from a `logon` when using Authentication and Entitlement.

Table B.6 contains the `AckType` messages which can be returned by a `logon` command.

Table B.6: Ack types supported by `logon`

ackType	Description
<code>none</code>	No <code>ack</code> message is returned. This is the default behavior.
<code>received</code>	The <code>logon</code> message has been received.
<code>persisted</code>	Not supported at this time.
<code>processed</code>	AMPS has processed the <code>logon</code> message(es).

Continued on next page

Table B.6 -- continued from previous page

ackType	Description
completed	Not supported at this time.
stats	Not supported at this time.

B.4 publish

Description

The `publish` command is the primary way to inject messages into the AMPS processing stream. A `publish` command received by AMPS will be forwarded to other connected clients with matching subscriptions.

Header Fields

Table B.7 contains the header fields available to a `publish` command.

Table B.7: Header fields supported by publish

Field	Description
Command	Command to be executed. Value: <code>publish</code>
Topic	Topic to place a subscription with.
AckType	Acknowledgment type for the given command. Value is a comma separated list of one or more of the following: <code>none</code> , <code>received</code> , <code>persisted</code> or <code>processed</code>
CommandId	If specified with an AMPS command requesting an <code>ack</code> , all requested <code>ack</code> messages will contain the <code>CommandId</code> in the <code>ack</code> response header.
Expiration	An <i>interval</i> , in seconds, used to define the lifetime of a <code>publish</code> message.
SequenceId	A monotonically increasing identifier used in high availability configurations to determine message uniqueness across replicas.
TransmissionTime	An ISO-8601 datetime used to note the time the message is sent by client.

Returns

A client which issues a `publish` can request a `processed` acknowledgment message, however this is not recommended as there is a significant performance overhead associated with this. Table B.8 contains the `AckType` messages which can be returned by a `publish`.

Table B.8: Ack types supported by `publish`

acks	Description
none	No <code>ack</code> message is returned. This is the default behavior.
received	AMPS has received the <code>publish</code> message.
persisted	When an AMPS engine returns an <code>ackType</code> of <code>persisted</code> this guarantees that 1) all downstream synchronous replication(s) all have acknowledged that the message(s) have been delivered to their respective SOW Topic(s). 2) when the <code>publish</code> message has been sent to all available downstream asynchronous replications.
processed	AMPS has processed the <code>publish</code> message.
completed	Not supported at this time.
stats	Not supported at this time.

Errors

Any errors that occur during this command will be returned in the status of a `processed` acknowledgment and logged to the log file. Regardless of success or failure, the `processed` acknowledgment will only be returned if requested by including `processed` in the `AckType` field.

Examples

Below are some examples of the `publish` command.

Publish command in XML

Here is an example of an XML message that when sent to AMPS will publish the FIXML message found in the SOAP body to all clients subscribed to the order topic.

```

1 <?xml version="1.0" encoding="iso-8859-1"?>
2 <SOAP-ENV:Envelope>
3   <SOAP-ENV:Header>

```

```
4      <Cmd>publish</Cmd>
5      <TxmTm>20061201-17:29:12.000-0500</TxmTm>
6      <Tpc>order</Tpc>
7  </SOAP-ENV:Header>
8  <SOAP-ENV:Body>
9      <FIXML>
10         <Order Side="2" Px="32.00"><Instrmt Sym="MSFT"/><
11             OrdQty Qty="100"/></Order>
12     </FIXML>
13 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Listing B.1: SOAP publish example

Publish command in FIX

Here is an example of a FIX publish message that uses ";" as the field separator, "|" as the header separator. This message will be sent to all subscribers of the order topic.

```
20000=publish;20005=order;|109=clientABCD;35=D;55=MSFT;
```

B.5 sow_and_delta_subscribe

Description

A `sow_and_delta_subscribe` command is used to combine the functionality of `sow` and a `delta_subscribe` command in a single command.

The `sow_and_delta_subscribe` command is used to query the contents of a SOW topic (this is the `sow` command) and place a subscription such that any messages which match the subscribed SOW topic and query filter will be published to the AMPS client (this is the `delta_subscribe` command). As with the `delta_subscribe` command, `publish` messages which are updates SOW records will only contain information which has changed.

If a `sow_and_delta_subscribe` is issued on a record which does not currently exist in the SOW topic, or if it is used on a topic which does not have a SOW-topic store defined, then a `sow_and_delta_subscribe` behaves like a `sow_and_subscribe` command.

Header Fields

Table B.9 contains the header fields supported by a `sow_and_delta_subscribe` command.

Table B.9: Header fields supported by `sow_and_delta_subscribe`.

Field	Description
Command	Command to be executed. Value: <code>sow_and_delta_subscribe</code>
Topic	The SOW topic to query and subscribe to.
AckType	Acknowledgment type for the given command. Value is a comma separated string of one or more of the following: <code>none</code> , <code>received</code> , <code>processed</code> , <code>completed</code> or <code>stats</code> .
BatchSize	Number of records to return in a single <code>sow</code> query result message. While the default value is 1, it is recommended to use a higher <code>/hdrFieldBatchSize</code> value, as even small increases can yield greater performance in query result delivery.
CommandId	If specified with an AMPS command requesting an <code>ack</code> , all requested <code>ack</code> messages will contain the <code>CommandId</code> in the <code>ack</code> response header.
DataOnly	Only send raw data to subscriber for a matching publish message if <code>true</code> . Removes SOAP envelope.
Filter	Content filter expression.
QueryId	Identifier used to identify the client's SOW topic query. This identifier will be added to all messages which are a response to the <code>sow_and_delta_subscribe</code>
SendEmpty	If set to <code>true</code> , empty published messages are forwarded to matching subscriptions. Default is <code>true</code> .
SendOOF	Messages which have fallen out of focus from the subscription are sent to the client. Default is <code>false</code> .
SendKeys	Option to instruct AMPS that the client would like to receive the <code>SowKey(s)</code> back.
QueryInterval	Query Interval that can be specified with a query and will result in the query being executed with a specified periodicity.
SendSubscriptionIds	If true, subscription identifiers will be sent for a matched message.
SubscriptionId	Identifier used to identify the client's subscription. If no subscription is provided as part of the <code>sow_and_delta_subscribe</code> then AMPS will generate one.
TransmissionTime	An ISO-8601 datetime used to note the time the message is sent by the client.

Returns

AMPS will send acknowledgment messages for the following `AckType` fields: `received`, `processed` along with a populated `Status` header field describing the acknowledgment.

If the `sow_and_delta_subscribe` command is successful, AMPS will return a `group_begin` message to notify the client that a group of messages is being returned. Chapter 5 will provide more information about SOW topic query behavior. Table B.10 contains the `AckType` messages which can be returned by a `sow_and_delta_subscribe`.

Table B.10: ack types supported by `sow_and_delta_subscribe`

ackType	Description
<code>none</code>	No ack message is returned. This is the default behavior.
<code>received</code>	The <code>sow_and_delta_subscribe</code> message has been received.
<code>persisted</code>	Not supported at this time.
<code>processed</code>	AMPS has compiled the filter(s) for the <code>sow_and_delta_subscribe</code> message(es).
<code>completed</code>	The <code>sow_and_delta_subscribe</code> message has completed the SOW portion of the command, and all future messages will be updated based on publishes.
<code>stats</code>	Returns an ack message with <code>Matches</code> , <code>TopicMatches</code> and <code>RecordsReturned</code> .

The `stats` message include three values in the header, the `Matches`, `TopicMatches` and the `RecordsReturned`. These are defined below:

TopicMatches The total number of records compared across all matching SOW topics.

Matches The number of records returned that match the topic regular expression and the content filter. This value can be greater than `RecordsReturned` in the case where the number of returned records is limited by `TopN`.

RecordReturned The total number of records returned to the client, which can be limited by the `TopN` header value.

Errors

Errors for a `sow_and_delta_subscribe` query are either returned in the `Status` field if an `AckType` has been defined, or the errors may be inserted into the AMPS log.

B.6 sow_and_subscribe

Description

A `sow_and_subscribe` command is used to combine the functionality of `sow` and a `subscribe` command in a single command. The `sow_and_subscribe` command is used to query the contents of a SOW topic (this is the `sow` command) and place a subscription such that any messages which match the subscribed SOW topic and query filter will be published to the AMPS client (this is the `subscribe` command).

Table B.11: Header fields supported by `sow_and_subscribe`.

Field	Description
Command	Command to be executed. Value: <code>sow_and_subscribe</code>
Topic	The SOW topic to query and subscribe to.
AckType	Acknowledgment type for the given command. Value is a comma separated string of one or more of the following: <code>none</code> , <code>received</code> or <code>processed</code> , <code>completed</code> or <code>stats</code> .
BatchSize	Number of records to return in a single <code>sow</code> query result message. While the default value is 1, it is recommended to use a higher <code>BatchSize</code> value, as even small increases can yield greater performance in query result delivery.
CommandId	If specified with an AMPS command requesting an <code>ack</code> , all requested <code>ack</code> messages will contain the <code>CommandId</code> in the <code>ack</code> response header.
DataOnly	Only send raw data to subscriber for a matching publish message if <code>true</code> . Removes SOAP envelope.
Filter	Content filter expression.
Options	A comma separated list of flags available to the <code>sow_and_subscribe</code> command. Table B.13 describes the <code>Options</code> available for use in the <code>subscribe</code> command.
QueryId	Identifier used to identify the client's SOW topic query. This identifier will be added to all messages which are a response to the <code>sow_and_subscribe</code>
QueryInterval	Query Interval that can be specified with a query and will result in the query being executed with a specified periodicity.
SendSubscriptionIds	If <code>true</code> , subscription identifiers will be sent for a matched message.

Continued on next page

TableB.11 -- continued from previous page

Field	Description
SubscriptionId	Identifier used to identify the client's subscription. If no subscription is provided as part of the <code>sow_and_subscribe</code> then AMPS will generate one.
TransmissionTime	An ISO-8601 datetime used to note the time the message is sent by the client.

Returns

AMPS will send acknowledgment messages for the following `AckType` fields: `received`, `processed` along with a populated `Status` header field describing the acknowledgment.

If the `sow_and_subscribe` command is successful, AMPS will return a `group_begin` message to notify the client that a group of messages is being returned. The [SOW Queries](#) chapter will give more information about SOW topic query behavior.

Table B.12 contains the `AckType` messages which can be returned by a `sow_and_subscribe`.

Table B.12: ack types supported by `sow_and_subscribe`

ackType	Description
<code>none</code>	No ack message is returned. This is the default behavior.
<code>received</code>	The <code>sow_and_subscribe</code> message has been received.
<code>persisted</code>	Not supported at this time.
<code>processed</code>	AMPS has compiled the filters for the <code>sow_and_subscribe</code> message(es).
<code>completed</code>	The <code>sow_and_subscribe</code> message has completed the SOW portion of the command, and all future messages will be updates based on publishes.
<code>stats</code>	Returns an ack message with <code>Matches</code> , <code>TopicMatches</code> and <code>RecordsReturned</code> .

Table B.13 contains a list of the `Options` available and their definitions when used in the AMPS command.

Table B.13: `Options` types supported by `subscribe`

Option	Description
<code>none</code>	This is the default <code>Options</code> type.

Continued on next page

TableB.13 -- continued from previous page

Option	Description
live	Tells AMPS to send messages to subscribing clients before they have been persisted to a downstream transaction log. The Subscribing for High Availability chapter contains a description of how a <code>live</code> subscription works in AMPS.
no_empties	Tells AMPS not to send empty publish messages to matching subscriptions.
oof	Send an OOF message for messages which have fallen out of focus from the original subscription.
replace	Replace the subscription associated with <code>CmdId</code> with another subscription.
send_keys	AMPS will send the SOW keys back with matching messages from the SOW.

The `stats` message include three values in the header, the `Matches`, `TopicMatches` and the `RecordsReturned`. These are defined below:

TopicMatches The total number of records compared across all matching SOW topics.

Matches The number of records returned that match the topic regular expression and the content filter. This value can be greater than `RecordsReturned` in the case where the number of returned records is limited by `TopN`.

RecordReturned The total number of records returned to the client, which can be limited by the `TopN` header value.

Errors

Errors for a `sow_and_subscribe` query are either returned in the `Status` field if an `AckType` has been defined, or the errors may be inserted into the AMPS log.

B.7 sow_delete

Description

In AMPS, there are three different ways to remove records from the SOW. The first method is to construct a `publish` message identical to the original message, then change the `Command` field to a `sow_delete` message. This has the net effect of causing AMPS recreate the `SowKey` for the particular message, then look up the `SowKey` message in the SOW and finally remove it.

The other method to remove messages from the SOW is to construct a `sow_delete` message and pass in a comma separated list of `SowKeys` in the message header which will cause all of the messages to be removed from the SOW Topic.

The third way to remove records from the SOW is similar to the manner in which a `sow` query command with a `filter` is performed. In this case, instead of returning the results of the `sow` command, those records which match the filter will be deleted from the SOW.

Header Fields

Table B.14 contains the header fields supported by a `sow_delete`.

Table B.14: Header fields supported by `sow_delete`.

Field	Description
Command	Command to be executed. Value: <code>sow_delete</code> .
Topic	The SOW Topic from which to delete the message(s) from.
AckType	Acknowledgment type for the given command. Value is a comma separated list of one or more of the following: <code>none</code> , <code>received</code> , <code>processed</code> , <code>persisted</code> , <code>completed</code> and <code>stats</code>
CommandId	If specified with an AMPS command requesting an <code>ack</code> , all requested <code>ack</code> messages will contain the <code>CommandId</code> in the <code>ack</code> response header.
SowKeys	A comma separated list of unique ids to be removed.
Filter	Content filter expression.



The `SowKeys` and `Filter` header fields can not be used together. They are mutually exclusive. Using them together in the same `sow_delete` command will cause indeterminate results.

Returns

For a `sow_delete` message, AMPS will send acknowledgment message, `completed` and `stats` for the following `AckType` fields: `received`, `processed` and `persisted` along with a populated `Status` header field describing the acknowledgment.

Table B.15: ack types supported by `sow_delete`

ackType	Description
none	No ack message is returned. This is the default behavior.
received	The <code>sow_delete</code> message has been received.
persisted	When an AMPS engine returns an <code>ackType</code> of <code>persisted</code> this guarantees that 1) all downstream synchronous replication(s) all have acknowledged that the message(s) have been deleted from their respective SOW Topic(s). 2) the <code>sow_delete</code> message has been sent to all available downstream asynchronous replications.
processed	AMPS has compiled the filter(s) for the <code>sow_delete</code> message(es).
completed	Supported for a <code>sow_delete</code> with a filter. The <code>completed</code> ack is returned when the query has completed.
stats	Returns an ack message with <code>Matches</code> , <code>TopicMatches</code> and <code>RecordsDeleted</code> .

The `stats` message include three values in the header, the `Matches`, `TopicMatches` and the `RecordsReturned`. These are defined below:

Errors

Errors which occur during a `sow_delete` are returned as part of the `processed` `AckType` message and recorded to the log. Typical errors involved a missing topic, or a missing/invalid `SowKey`

Examples

Below are some examples of how to use the `sow_delete` command in AMPS using both the FIX message format and the SOAP XML message format.

FIX `sow_delete` examples

In the first example, the `sow_delete` command will delete the records with `SowKey` equal to `c176c6,m2210c4` and `c0706a34`. Each of the records will be listed as a comma-separated items in the `SowKeys` header field.

```
20000=sow_delete;20005=order;20032=c176c6,m2210c4,↵
c0706a34;20008=processed;
```

In the next example, a FIX message will be constructed which will delete the message which was published in [Section B.4](#). Like that message, the ";" character will be used as the field separator and the "|" character will be used as the header separator. This message will delete the previously published message from the all SOW topic and any downstream replicas where the message may have been persisted.

```
20000=sow_delete;20005=order;|109=clientABCD;35=D;55=↵
MSFT;
```

XML sow_delete example

Similar to the previous FIX example, this SOAP message constructed for AMPS will delete the message published in the example listed in [Section B.4](#).

```
1 <?xml version="1.0" encoding="iso-8859-1"?>
2 <SOAP-ENV:Envelope>
3   <SOAP-ENV:Header>
4     <Cmd>sow_delete</Cmd>
5     <TxmTm>20121201-12:11:01.000-0500</TxmTm>
6     <Tpc>order</Tpc>
7   </SOAP-ENV:Header>
8   <SOAP-ENV:Body>
9     <FIXML>
10      <Order Side="2" Px="32.00"><Instrmt Sym="MSFT"/><
11        OrdQty Qty="100"/></Order>
12    </FIXML>
13  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Listing B.2: SOAP sow_delete example

Notes

Never combine the use of the `SowKeys` header method with the `SowKey` body method. AMPS will always examine the `SowKeys` header message first and delete those messages regardless of the contents of the message body.

B.8 sow

Description

The `sow` command is used to query the contents of a previously defined SOW Topic. A `sow` command can be used to query an entire SOW Topic, or a filter can be used to further refine the results found inside a SOW Topic. See the [State of the World \(SOW\)](#) and [SOW Queries](#) chapters for more information.

Header Fields

Table B.16 contains the header fields supported by the `sow` command.

Table B.16: Header fields supported by `sow`.

Field	Description
Command	Command to be executed. Value: <code>sow</code> .
Topic	The SOW Topic from which the records will be queried.
AckType	Acknowledgment type for the given command. Value is a comma separated list of one or more of the following: <code>none</code> , <code>received</code> , <code>processed</code> , <code>completed</code> or <code>stats</code> .
CommandId	If specified with an AMPS command requesting an <code>ack</code> , all requested <code>ack</code> messages will contain the <code>CommandId</code> in the <code>ack</code> response header.
QueryId	Unique identifier which is returned as part of the response delivered back to the client.
Filter	See the Content Filtering chapter for more information.
BatchSize	Number of records to return in a single <code>sow</code> query result message. While the default value is 1, it is recommended to use a higher <code>BatchSize</code> value, as even small increases can yield greater performance in query result delivery.

Returns

When a `sow` message is received AMPS can return a `received` message as notification that the message has arrived. When the message has been processed, AMPS will return the `processed` command along with the errors if any have occurred. Otherwise the `processed` will be an acknowledgment.

The results returned by a SOW are put into a `sow` record group by first sending a `group_begin` message, followed by the matching SOW records. A `group_end` message is used to denote the close of query results processing.

Table B.17 contains a listing of the acknowledgment messages supported by the `sow` command.

Table B.17: `ack` types supported by `sow`

<code>ackType</code>	Description
<code>none</code>	No <code>ack</code> message is returned. This is the default behavior.

Continued on next page

TableB.17 -- continued from previous page

ackType	Description
received	The <code>sow</code> message has been received.
persisted	Not supported at this time.
processed	AMPS has compiled the filter(s) for the <code>sow</code> message(es).
completed	The <code>sow</code> message has completed the <code>SOW</code> portion of the command.
stats	Returns statistics related to the state of the <code>SOW</code> query results.

The `stats` message include three values in the header, the `Matches`, `TopicMatches` and the `RecordsReturned`. These are defined below:

TopicMatches The total number of records compared across all matching `SOW` topics.

Matches The number of records returned that match the topic regular expression and the content filter. This value can be greater than `RecordsReturned` in the case where the number of returned records is limited by `TopN`.

RecordReturned The total number of records returned to the client, which can be limited by the `TopN` header value.



The ordering of records returned by a `SOW` query is undefined.

B.9 start_timer

Description

The `start_timer` resets the AMPS client and latency statistics counter, and starts the timer for for the client and latency statistic counter. This is used in conjunction with `stop_timer` to generate statistics which are published to the AMPS log.

Header Fields

Table B.18 contains the header fields supported by the `start_timer` command.

Table B.18: Header fields supported by `start_timer`.

Field	Description
Command	Command to be executed. Value: <code>start_timer</code>

B.10 `stop_timer`

Description

The `stop_timer` command is used in conjunction with the `start_timer`. Once a `stop_timer` command has been issued, AMPS will write statistics to the log file to generate a profile of the number of clients and messages observed and processed by AMPS.

Header Fields

Table B.19 contains the header fields supported by the `stop_timer` command.

Table B.19: Header fields supported by `stop_timer`.

Field	Description
Command	Command to be executed. Value: <code>stop_timer</code>
AckType	Acknowledgment type for the given command. Only <code>processed</code> is supported for this command.
CommandId	If specified with an AMPS command requesting an <code>ack</code> , all requested <code>ack</code> messages will contain the <code>CommandId</code> in the <code>ack</code> response header.

Returns

The `stop_timer` command can return an acknowledgment message if the `AckType` is set to `processed`. The returned `ack` message will contain a `Status` field which will contain any messages from AMPS.

The `stop_timer` will return a `ClientStatus` message which contains the statistics measured since the last call to the `start_timer` command. Table B.20 contains the `AckType` messages supported by the `stop_timer` command.

Continued on next page

TableB.20 -- continued from previous page

ackType	Description
Table B.20: ack types supported by <code>sow</code>	
ackType	Description
<code>none</code>	Not supported at this time.
<code>received</code>	Not supported at this time.
<code>parsed</code>	Not supported at this time.
<code>persisted</code>	Not supported at this time.
<code>processed</code>	AMPS has processed the <code>stop_timer</code> message(es).
<code>completed</code>	Not supported at this time.
<code>stats</code>	Not supported at this time.

B.11 subscribe

Description

The `subscribe` command is the primary way to retrieve messages from the AMPS processing stream. A client can issue a `subscribe` command on a topic to receive all published messages to that topic in the future. Additionally, content filtering can be used to choose which messages the client is interested in receiving.

Header Fields

Table B.21 contains the supported header fields for the `subscribe` command.

Table B.21: Header fields supported by `subscribe`.

Field	Description
Command	Command to be executed. Value: <code>subscribe</code>
Topic	Topic to place a subscription with.
AckType	Acknowledgment type for the given command. Value is a comma separated list of one or more of the following: <code>none</code> , <code>received</code> , <code>processed</code> , or <code>completed</code> . Table B.22 describes the acknowledgement messages.
CommandId	If specified with an AMPS command requesting an <code>ack</code> , all requested <code>ack</code> messages will contain the <code>CommandId</code> in the <code>ack</code> response header.

Continued on next page

Table B.21 -- continued from previous page

Field	Description
DataOnly	A Boolean value (<code>true</code> or <code>false</code>) which, if <code>true</code> , will only send raw data to subscriber for a matching publish message i.e., do not include SOAP envelope.
Filter	DATA wrapped string which is used as a content filter expression.
Options	A comma separated list of flags available to the <code>subscribe</code> command. Table B.23 describes the <code>Options</code> available for use in the <code>subscribe</code> command.
SendSubscriptionIds	Boolean (<code>true</code> or <code>false</code>) subscription identifiers will not be sent for a matched message.
TransmissionTime	An ISO-8601 datetime used to note the time the message is sent by client.

Returns

It is possible to specify an `processed` acknowledgment be sent back to the client that issued the `subscribe` command. Within this `processed` acknowledgment, a client can get back the result of placing the subscription (success or failure) and the `SubscriptionId` which uniquely identifies the subscription within AMPS. Keeping track of the `SubscriptionId` is useful for unsubscribing from subscriptions and issuing `SOW` queries.

[Table B.22](#) contains a list of the supported `AckType` messages available to the `subscribe` command.

Table B.22: ack types supported by `subscribe`

Ack Type	Description
<code>none</code>	No <code>ack</code> message is returned. This is the default behavior.
<code>received</code>	The <code>subscribe</code> message has been received.
<code>persisted</code>	Not supported at this time.
<code>processed</code>	AMPS has compiled the filter(s) for the <code>subscribe</code> message(es).
<code>completed</code>	The <code>subscribe</code> message has completed the <code>SOW</code> portion of the command, and all future messages will be updates based on publishes.

[Table B.23](#) contains a list of the of `Options` available and their definitions in the AMPS command.

Table B.23: Options types supported by subscribe

Option	Description
none	This is the default Options type.
live	Tells AMPS to send messages to subscribing clients before they have been persisted to a downstream transaction log. The Subscribing for High Availability chapter contains a description of how a live subscription works in AMPS.
no_empties	Not supported by this message type.
oof	Not supported by this message type.
replace	Replace the subscription associated with CmdId with another subscription.
send_keys	Not supported by this message type.

Errors

Any errors that occur during this command will be returned in the status of a processed acknowledgment and logged to the log file. Regardless of success or failure, the processed acknowledgment will only be returned if requested by including processed in the AckType field.

Examples

Below are some examples of the subscribe command.

subscribe command in XML

Here is an example of an XML message that when sent to AMPS will subscribe to all messages on the order topic that match the following filter:

```
{/FIXML/Order/Instrmt/@Sym = 'IBM') AND ←
  (/FIXML/Order/@Px >= 90.00 AND
 /FIXML/Order/@Px < 91.0)}
```

```
1 <?xml version="1.0" encoding="iso-8859-1"?>
2 <SOAP-ENV:Envelope>
3   <SOAP-ENV:Header>
4     <Cmd>subscribe</Cmd>
5     <TxmTm>20061201-17:29:12.000-0500</TxmTm>
6     <Tpc>order</Tpc>
7     <AckTyp>processed</AckTyp>
8     <Fltr>
9       <![CDATA[ (/FIXML/Order/Instrmt/@Sym =
10        'IBM') AND (/FIXML/Order/@Px >= 90.00 AND
```

```

11      /FIXML/Order/@Px < 91.0)]]>
12      </Fltr>
13      </SOAP-ENV:Header>
14      <SOAP-ENV:Body />
15 </SOAP-ENV:Envelope>

```

B.12 unsubscribe

Description

The `unsubscribe` command allows a client to notify amps that it no longer wishes to receive messages related to a previous subscription. A client can issue a `unsubscribe` for an individual subscription by using the `SubId` returned from the `subscribe` command, or the user may use the `all` keyword to unsubscribe from all AMPS SOW topic subscriptions.

Header Fields

Table B.24 contains the header fields supported by `unsubscribe`.

Table B.24: Header fields supported by `unsubscribe`.

Field	Description
Command	Command to be executed. Value: <code>unsubscribe</code>
SubId	Subscription Id returned from AMPS to the client when the original <code>subscription</code> was placed. The keyword <code>all</code> can also be used to unsubscribe from all current subscriptions for the client.
AckTyp	Acknowledgment type for the given command. Value is a comma separated list of one or more of the following: <code>none</code> , <code>received</code> or <code>processed</code> .
CommandId	If specified with an AMPS command requesting an <code>ack</code> , all requested <code>ack</code> messages will contain the <code>CommandId</code> in the <code>ack</code> response header.

Returns

An `unsubscribe` command will acknowledge a `received` when the `unsubscribe` message is received and the `AckTyp` is set to `received`. AMPS will also return an `ack` for a `processed` request.

Table B.25 contains a the supported `AckType` messages available to the `unsubscribe` command.

Table B.25: ack types supported by `unsubscribe`

ackType	Description
<code>none</code>	No <code>ack</code> message is returned. This is the default behavior.
<code>received</code>	The <code>unsubscribe</code> message has been received.
<code>persisted</code>	Not supported at this time.
<code>processed</code>	AMPS has processed the <code>unsubscribe</code> message(es).
<code>completed</code>	Not supported at this time.
<code>stats</code>	Not supported at this time.

Appendix **C**

Configuration Reference

Once you have reviewed the [Getting Started](#) chapter and would like to start configuring some of the more advanced features of AMPS, you will want to keep this chapter handy. If you have not read [Chapter 2](#) for the First Steps for AMPS, please start there before reading this chapter.

C.1 AMPS Configuration Basics

The easiest way to create a custom XML configuration file for AMPS is to use the `amps_config.xml` file provided in the `$AMPSDIR/demo` directory. [Listing C.1](#) shows a simplified sample configuration file.

```
1 <?xml version="1.0"?>
2 <AMPSConfig>
3
4   <Name>AMPS</Name>
5
6   <Admin>
7     <InetAddr>localhost:9090</InetAddr>
8     <FileName>./stats.db</FileName>
9   </Admin>
10
11  <Transports>
12    <Transport>
13      <Name>fix-tcp</Name>
14      <Type>tcp</Type>
15      <InetAddr>9004</InetAddr>
16      <ReuseAddr>false</ReuseAddr>
17      <MessageType>fix</MessageType>
18    </Transport>
19    <Transport>
```

```
20 <Name>xml-tcp</Name>
21 <Type>tcp</Type>
22 <InetAddr>9005</InetAddr>
23 <ReuseAddr>>false</ReuseAddr>
24 <MessageType>xml</MessageType>
25 </Transport>
26 </Transports>
27
28 <Logging>
29 <Target>
30 <Protocol>stdout</Protocol>
31 <Level>errors</Level>
32 <Levels>stats</Levels>
33 <ExcludeErrors>05-0006</ExcludeErrors>
34 </Target>
35 </Logging>
36
37 </AMPSConfig>
```

Listing C.1: Simple AMPS configuration file

The AMPS configuration XML file is defined first by wrapping the config file with an `AMPSConfig` tag to identify it as a configuration file. Next, the instance is given a name using the `<Name>` tag.

Once our instance has a name, it is good to define where the administration port will be connecting to. By default, the administration port can be found by pointing a browser to `http://localhost:8085`, but if a different port or host name is desired, then that is defined in the `Admin` and `InetAddr` tags. The `Admin` port is discussed more in [Table C.5](#).

Next up is how to get messages into AMPS. There are several different transport types which can be parsed by AMPS, all of which are discussed in greater detail in the [Transports](#) chapter, but for now this chapter will focus on the two most common, `fix-tcp` and `xml-tcp`. In AMPS, each key to defining each transport is to give them a unique `InetAddr` port and a unique `MessageType` tag. This pairing is how AMPS is able to determine which parser to apply to different incoming messages on a specific port. In the above example, if messages are coming in on port 9005, then the xml parser will be used to parse and extract the message data. Similarly, if messages are received on port 9004, then a FIX parser will be used to parse and extract message data.

The last portion of the configuration is Logging. In the above example, the `Logging` tag defines only one log target, however one or more `Logging` targets is quite common. Again referring to the example, all logging messages which are errors and below, in addition to all log messages which are `stats` will be logged to the `Protocol` definition of `stdout` - or in other words, these messages will be logged to the terminal, and not to a file. AMPS supports a robust set of logging features and configurations, all of which are covered in more detail in the [Logging](#) chapter.

C.1.1 AMPS Configuration File Special Characters

Log Rotation Name

When specifying an AMPS log file which has `RotationThreshold` specified, using the `%n` string in the log file name is a useful mechanism for ensuring the name of the log file is unique and sequential. [Listing C.2](#) shows a file name token replacement in the AMPS configuration file.

```
1 <Logging>
2   <Target>
3     <Protocol>file</Protocol>
4     <Level>info</Level>
5     <FileName>log/log-%n.log</FileName>
6     <RotationThreshold>2G</RotationThreshold>
7   </Target>
8 </Logging>
```

Listing C.2: File tokens used in configuration file

In the above example, a log file will be created in the `AMPSDIR/log/` directory. The first time this file is created, it will be named `log-1.log`. Once the log file reaches the `RotationThreshold` limit of 2G, the previous log file will be saved, and the new log file name will be incremented by one. Thus, the next log file will be named `AMPSDIR/log/log-2.log`.

Dates

AMPS allows administrators to use date based file names when specifying the file name in the configuration, as demonstrated in [Listing C.3](#).

```
1 <Logging>
2   <Target>
3     <Protocol>file</Protocol>
4     <Level>info</Level>
5     <FileName>
6       log/log-%Y-%m-%dT%H%M%S.log
7     </FileName>
8     <RotationThreshold>2G</RotationThreshold>
9   </Target>
10 </Logging>
```

Listing C.3: Date tokens used in configuration file

In the above example, a log file will be created in the `$AMPSDIR/log` named `2011-01-01-120000.log` if the log was created at noon on January 1, 2011.

C.1.2 Using Units in the Configuration

To make configuration easy, AMPS permits the use of units to expand values. For example, if a time interval is measured in seconds, the letter *s* can be appended to the value. For example, the following SOW topic definition used the `Expiration` tag to set the record expiration to 86400 seconds (one day).

```

1 <TopicDefinition>
2   ...
3   <Expiration> 86400s </Expiration>
4   ...
5 </TopicDefinition>

```

Listing C.4: Expiration using seconds.

An even easier way to specify an expiration of one day is to use the following `Expiration`:

```

1 <TopicDefinition>
2   ...
3   <Expiration>1d</Expiration>
4   ...
5 </TopicDefinition>

```

Listing C.5: Expiration using seconds.

Table C.1 shows a listing of the time units AMPS supports in the configuration file.

Table C.1: AMPS Configuration - Time Units

Units	Description
ns	nanoseconds
us	microseconds
ms	milliseconds
s	seconds
m	minutes
h	hours
d	days
w	weeks

AMPS configuration supports a similar mechanism for byte-based units when specifying sizes in the configuration file, also. Table C.2 shows a listing of the byte units AMPS supports in the configuration file.

Table C.2: AMPS Configuration - Byte Units

Units	Description
kb	kilobytes
mb	megabytes
gb	gigabytes
tb	terabytes

Dealing with large numbers in AMPS configuration can also be simplified by using common exponent values to handle raw values. This means that instead of having to input 10000000 to represent ten million, a user can input 10M. [Table C.3](#) contains a list of the exponents supported.

Table C.3: AMPS Configuration - Numeric Units

Units	Description
k	10 ³ - thousand
M	10 ⁶ - million



To make remembering the units easier, AMPS interval and byte units are not case sensitive.

C.1.3 Environment Variables in AMPS Configuration

AMPS configuration also allows for environment variables to be used as part of the data when specifying a configuration file.

If a global system variable is commonly used in an organization, it may be useful to define this in one location and re-use it across multiple AMPS installations or applications. AMPS will replace any token which is wrapped with `${}` with the environment variable which defined in the current user operating system environment. [Listing C.6](#) demonstrates how the environment variable `ENV_LOG` is used to define a global environment variable for the location of the host logging.

```

1 <Logging>
2   <Target>
3     <Protocol>file</Protocol>
4     <FileName>${ENV_LOG}</FileName>
5     <Level>info</Level>
6     <RotationThreshold>2G</RotationThreshold>
7   </Target>

```



```
8 </Logging>
```

Listing C.6: Environment variable used in configuration

Internal Environment Variables

In addition to supporting custom environment variables, AMPS includes a configuration variable, `AMPS_CONFIG_DIRECTORY`, which can be used to reference the directory in which the configuration file used to start AMPS is located. For example if AMPS was started with the following command at the command prompt:

```
%> ./ampServer ../amps/config/config.xml
```

Then the log file configuration option shown in [Listing C.7](#) can be used to instruct AMPS to create the log files in the same parent directory as the configuration file -- in this case `../amps/config/logs/infoLog.log`.

```
1 <Logging>
2 <Target>
3   <Protocol>file</Protocol>
4   <FileName>
5     ${AMPS_CONFIG_DIRECTORY}/logs/infoLog.log
6   </FileName>
7   <Level>info</Level>
8   <RotationThreshold>2G</RotationThreshold>
9 </Target>
10 </Logging>
```

Listing C.7: `AMPS_CONFIG_DIRECTORY` Environment variable example

In addition to the `AMPS_CONFIG_DIRECTORY` environment variable, AMPS also supports the `AMPS_CONFIG_PATH` which is an absolute path to the configuration file used to start AMPS.

C.2 Generating a Configuration File

This appendix includes a listing of all AMPS configuration parameters. AMPS provides a command line option to help an administrator quickly set up an AMPS server. In addition to the quick setup discussed in the [Next Steps](#) section of the [Getting Started](#) chapter, AMPS also provides the following command line options to create a basic XML configuration file. Running the following command will create a configuration file named `config.xml` which will be used as a starting point to discuss configuration in this appendix.

```
ampServer --sample-config > config.xml
```

C.3 Features

Below we will be breaking down each of the tags and defining their respective roles in configuring the AMPS server.

C.3.1 Name

The `Name` element give the AMPS instance a name.

Table C.4: Name Parameters

Element	Description
Name	This element defines the name of your AMPS instance. The name is used as the 'ident' parameter when logging to syslog

```
1 <Name>AMPS</Name>
```

Listing C.8: Name Example

C.3.2 Admin

The `Admin` tag is used to control the behavior of the administration server.

Table C.5: Admin Parameters

Element	Description
InetAddr	Defines a port for the embedded HTTP admin server, which can then be accessed via a browser
FileName	location for storing the statistics information reported by the Admin Server default: <code>:memory:</code> - location to store statistics information.
Interval	The refresh interval for the Admin Server to update gathered statistics default: <code>10s</code> minimum: <code>1s</code>

```
1 <Admin>
2   <InetAddr>localhost:9090</InetAddr>
3   <FileName>stats.db</FileName>
4   <Interval>20s</Interval>
5 </Admin>
```

Listing C.9: Admin Example

C.3.3 Modules

The `Modules` section of the AMPS configuration file is used to configure and define any plug-in modules which have been written for AMPS. The modules supported for AMPS are the authorization and entitlements modules.

The available features of a `Module` are listed in [Table C.6](#).

Table C.6: Module Parameters

Element	Description
Name	A plain text name for the module. This will be used as a reference when the module is implemented in concert with another AMPS feature (for example, a SOW topic).
Library	The location of the module library which will be called to implement interface's feature.
Options	A list of supported features for the implemented library.

An example of an AMPS configuration which uses an authorization and entitlement plug-in module is listed below in [Listing C.10](#). In our example, a custom authentication module named `libauthenticate_customer001.so` has been written to manage the authentication portion of AMPS authentication. Similarly, a custom entitlements module has been written named `libentitlement_customer001.so` to manage the permissions and access of the authenticated user.

The first step is to define the global `Modules` section of the AMPS configuration, and then list the individual modules.

```

1 <AMPSConfig>
2   ...
3   <Modules>
4     <Module>
5       <Name>authentication1</Name>
6       <Library>libauthenticate_customer001.so</Library>
7       <Options>
8         <LogLevel>info</LogLevel>
9         <Mode>debugging</Mode>
10      </Options>
11     </Module>
12     <Module>
13       <Name>entitlement1</Name>

```

```

14     <Library>libentitlement_customer001.so</Library>
15     <Options>
16         <LogLevel>error</LogLevel>
17         <Mode>prod</Mode>
18     </Options>
19 </Module>
20 ...
21 </Modules>
22 ...
23 </AMPSConfig>

```

Listing C.10: Sample global config of authentication and entitlements modules.

We now have an authentication module and an entitlements module that we can reference elsewhere in the AMPS configuration file to enable authentication and/or entitlements for supported features. For example, we can globally list our `Authentication` as part of the AMPS configuration file and then use it to ensure that our `Transports` are properly enabling authentication and entitlements. This is accomplished via an entry similar to [Listing C.11](#).

```

1 <AMPSConfig>
2 ...
3 <Authentication>
4     <Module>amps-no-authorization</Module>
5 </Authentication>
6 <Entitlement>
7     <Module>amps-no-authorization</Module>
8 </Entitlement>
9 ...
10 <Transports>
11 <Transport>
12     <Name>fix-tcp-001</Name>
13     ...
14     <Authentication>
15         <Module>authenticate_customer001</Module>
16     </Authentication>
17     <Entitlement>
18         <Module>entitlement_customer001</Module>
19     </Entitlement>
20 </Transport>
21 <Transport>
22     <Name>fix-tcp-007</Name>
23     ...
24     <Authentication>
25         <Module>authenticate_customer007</Module>
26     </Authentication>
27     <Entitlement>
28         <Module>entitlement_customer007</Module>
29     </Entitlement>
30 </Transport>
31 </Transports>

```

```

32  . . .
33  </AMPSConfig>

```

Listing C.11: Example of security enabled transports.

In example listed in [Listing C.11](#), our `fix-tcp-001` transport is secured with the `authenticate_customer001` authentication module, and the `entitlement_customer001` entitlement module, which is defined in a global `Modules` section similar to the one listed in [Listing C.10](#). Similarly, the `fix-tcp-007` transport is secured with the `authenticate_customer007` authentication module and the `entitlement_customer007` entitlement module.

C.3.4 Message Types

This tag defines the message types supported by the AMPS instance. A single AMPS instance can support multiple message types, as such a `MessageTypes` can contain multiple `MessageType` definitions.

The `Type` attributes for `fix`, `nvfix`, `xml` and `soapxml` have already been defined in AMPS, so redefinition of them is only required if a `Type`'s settings need to be changed (i.e.: the `MessageSeparator` for a `fix` message).

Table C.7: Message Type Parameters

Element	Description
Name	This element defines the name for the message type. The name is used to specify <code>MessageType</code> in other sections, e.g. <code>Transport</code> , <code>TopicDefinition</code> .
Module	The element defines the 'dynamic library' module that will be loaded for this message type. Valid Modules are: <ul style="list-style-type: none"> • <code>fix</code>: Standard fix, only allow numeric tags • <code>nvfix</code>: Standard fix, but allows non-numeric tags • <code>soapfix</code>: Standard fix wrapped inside a SOAP body • <code>xml</code>: XML wrapped inside a SOAP body
Type	
FieldSeparator	Sequence of characters used to separate field items in a FIX message. Note: this field is the ASCII value of the char sequence.
HeaderSeparator	Sequence of characters used to separate the header from the body in a FIX message. Note: this field is the ASCII value of the char sequence.

Continued on next page

TableC.7 -- continued from previous page

Element	Description
MessageSeparator	Sequence of characters used to separate message items in the body in a FIX message. Note: this field is the ASCII value of the char sequence.
AMPSVersionCompliance	When set to 2, this flag adds the fields used in the <code>SOWStats</code> internal messages which were found in AMPS versions prior to the version 3.0 release.

```

1 <MessageTypes>
2 <MessageType>
3   <Name>fix</Name>
4   <Module>fix</Module>
5   <?- The following are FIX specific options ?>
6   <FieldSeparator>1</FieldSeparator>
7   <HeaderSeparator>2</HeaderSeparator>
8   <MessageSeparator>3</MessageSeparator>
9 </MessageType>
10 </MessageTypes>

```

Listing C.12: Message Types Example

C.3.5 Transports Types

AMPS supports FIX and XML message formats as the transport types for communication between publishers and subscribers. `Transports` is a container which contains a list one or more of `Transport` definitions. This section will focus on the singular `Transport`.

Table C.8: Transport Parameters

Element	Description
InetAddr	The port AMPS will listen on for this transport.
Name	Define a name for the Transport.
Type	The type of Transport valid values: tcp
MessageType	Defines the message type for this transport, and is a reference to the name of a specific message type defined in the <code>MessageTypes</code> section.
ReuseAddr	Permits an AMPS instance to use a socket which is in a WAIT state. This can occur when AMPS has been restarted using the same <code>InetAddr</code> and the previous instance did not fully close the port. - valid values: true, false - default: false

Continued on next page

TableC.8 -- continued from previous page

Element	Description
SlowClientDisconnect	Define whether or not to disconnect clients that are slow after they breach the offline threshold. By default <code>SlowClientDisconnect</code> is enabled and will disconnect - default: true - valid values: true, false
ClientOffline	Defines whether or not a slow client should offline messages to avoid unlimited memory queuing. - valid values: enabled, disabled - default: disabled
ClientBufferThreshold	Defines how much memory a slow client can use before off lining. - units: Bytes
ClientOfflineThreshold	Defines how many messages a slow client can offline before being disconnected. - units: Messages
ClientOfflineDirectory	Location to persist messages for a slow client in anticipation of it resuming message processing. Required if <code>ClientOffline</code> is enabled.

```

1 <Transports>
2
3 <!-- fix messages using TCP -->
4 <Transport>
5   <Name>fix-tcp</Name>
6   <Type>tcp</Type>
7   <InetAddr>9004</InetAddr>
8   <ReuseAddr>true</ReuseAddr>
9   <MessageType>fix</MessageType>
10  <ClientBufferThreshold>
11    4194304
12  </ClientBufferThreshold>
13  <ClientOffline>enabled</ClientOffline>
14  <ClientOfflineThreshold>
15    10000
16  </ClientOfflineThreshold>
17  <ClientOfflineDirectory>
18    /var/tmp
19  </ClientOfflineDirectory>
20  <SlowClientDisconnect>true</SlowClientDisconnect>
21 </Transport>
22
23 <!-- nvfix messages using TCP -->
24 <Transport>
25   <Name>nvfix-tcp</Name>
26   <Type>tcp</Type>

```

```

27     <InetAddr>9005</InetAddr>
28     <ReuseAddr>true</ReuseAddr>
29     <MessageType>nvfix</MessageType>
30 </Transport>
31
32 <!-- xml messages using TCP -->
33 <Transport>
34     <Name>soap-tcp</Name>
35     <Type>tcp</Type>
36     <InetAddr>9006</InetAddr>
37     <ReuseAddr>true</ReuseAddr>
38     <MessageType>xml</MessageType>
39 </Transport>
40
41 </Transports>

```

Listing C.13: Transports Example

C.3.6 Logging

AMPS supports several different types of log formats, and multiple targets can be defined simultaneously.

Table C.9: Logging Parameters

Element	Description
Protocol	Define the logging target protocol - valid values: stdout, stderr, file, gzip, syslog
FileName	File to log to. If RotationThreshold is specified then %n is added to the file. If the protocol is gzip, then .gz is added to the file name - default: \$PWD/%Y-%m-%dT%H%M%S.log
RotationThreshold	Log size at which log rotation will occur. See Table C.2 for details on specifying file size.
Level	Defines a lower bound (inclusive) log level at which all log messages at the specified level and up are logged. - valid values: none, trace, debug, stats, info, warning, error, critical, emergency
Levels	A comma separated list of specific log levels. Only log messages at the specified levels will be logged - valid values: none, trace, debug, stats, info, warning, error, critical, emergency
IncludeErrors	All errors that should be included when logging.
ExcludeErrors	All errors that should be excluded when logging.

Continued on next page

TableC.9 -- continued from previous page

Element	Description
Ident	Syslog identifier for the AMPS instance. - default: AMPS Instance Name
Options	A comma separated list of syslog options.
Facility	Syslog facility to use.

```

1 <Logging>
2   <Target>
3     <Protocol>file</Protocol>
4     <FileName>
5       /var/tmp/amps/logs/\%Y\%m\%d\%H\%M\%S-\%n.log
6     </FileName>
7     <RotationThreshold>2G</RotationThreshold>
8     <Level>trace</Level>
9     <Levels>critical</Levels>
10  </Target>
11  <Target>
12    <Protocol>syslog</Protocol>
13    <Level>critical</Level>
14    <Ident>amps_dma</Ident>
15    <Options>LOG_CONS,LOG_NDELAY,LOG_PID</Options>
16    <Facility>LOG_USER</Facility>
17  </Target>
18 </Logging>

```

Listing C.14: Logging Example

C.3.7 State of the World (SOW)

State of the World (SOW) provides a mechanism for AMPS to persist the most recent publish for each message.

Table C.10 contains a listing of the parameters for a `TopicDefinition` section in the `TopicMetaData` section of an AMPS configuration file.

Table C.10: State of the World Parameters - TopicDefinition

Element	Description
FileName	The file where the State of the World data will be stored.
Key	Defines the specific key (as an XPath) whose value makes a message unique. This determines when a new record is created versus updated. This element can be specified multiple times to create a composite key.

Continued on next page

TableC.10 -- continued from previous page

Element	Description
MessageType Topic	Type of messages to be stored. SOW topic - all unique messages (see <code>Key</code> above) on this topic will be stored in a topic specific SOW database.
RecordSize	Size (in bytes) of a SOW record for this topic. - default: 1024
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 (in seconds) for how long a record should live in the SOW database for this topic. This element is only honored when <code>ExpirationProcessing</code> is enabled. A value of 0 indicates that the records should never be expired.
KeyDomain	The seed value for <code>SowKeys</code> used with in the topic. The default is the topic name, but it can be changed to a string value to unify <code>SowKey</code> values between different topics.
Duration	Define the storage duration for a SOW topic. SOW databases listed as <code>persistent</code> retain their instance life cycles, while those listed as <code>transient</code> do not. -default: persistent -valid values: persistent,transient

An example of a SOW configuration looks like the following:

```

1 <TopicMetaData>
2   <TopicDefinition>
3     <Topic>orders</Topic>
4     <Name>orders</Name>
5     <Key>/orderId</Key>
6     <MessageType>nvfix</MessageType>
7     <FileName>./sow/%n</FileName>
8   </TopicDefinition>
9 </TopicMetaData>

```

Listing C.15: SOW TopicMetaData Configuration

C.3.8 Replication Destination

An AMPS replication target is defined within the `Replication` section of an AMPS configuration file. Within the `Replication` section, there are one or more `Destination` sections, each specifying a unique replication target. [Table C.11](#) contains a listing of the parameters for the `Destination` section in the `Replication` section of an AMPS configuration file.

Table C.11: State of the World Parameters - ReplicaDefinition

Element	Description
Destination	Required parent tag which defines a unique replication target.
SyncType	Defines how synchronization of ack messages is handled. Is one of <code>sync</code> or <code>async</code> .
Transport	The message type and URI where messages will be replicated. Requires a <code>Type</code> , <code>MessageType</code> which must be "amps-replication" and <code>InetAddr</code> .
Name	The name of the replication.
Topic	Defines the topic name for this replica. Requires a <code>Name</code> and <code>MessageType</code>

```

1 <Replication>
2   <Destination>
3     <Name>amps-2</Name>
4     <Topic>
5       <Name>ORDER_STATE-Replication</Name>
6       <MessageType>xml</MessageType>
7     </Topic>
8     <SyncType>sync</SyncType>
9     <Transport>
10      <MessageType>xml</MessageType>
11      <InetAddr>localhost:19005</InetAddr>
12    </Transport>
13  </Destination>
14 </Replication>

```

Listing C.16: Replication example

C.3.9 View Definition

[Table C.12](#) contains a listing of the parameters for a `ViewDefinition` section in the `TopicMetaData` section of an AMPS configuration file.

Table C.12: State of the World Parameters - ViewDefinition

Element	Description
FileName	File location to store view data.
MessageType	One of <code>fix</code> , <code>xml</code> or <code>nvfix</code> .
Topic	Defines the topic name for this view.
UnderlyingTopic	Defines the SOW topic this view is based on.
Projection/Field	Defines what the view will contain. This element can be specified multiple times to compose a complex view. Complex expressions that use aggregation functions and conditional branching can also be used.
Grouping/Field	Defines how the records in the underlying topic will be grouped. This is analogous to a 'group by'
KeyDomains	The seed value for SowKeys used within this topic. The default is the topic name, but it can be changed to a string value to unify SowKey values between different topics.

```

1
2 <TopicMetaData>
3   <TopicDefinition>
4     <Topic>/ett/order</Topic>
5     <MessageType>fix</MessageType>
6     <Key>/orderId</Key>
7   </TopicDefinition>
8   <ViewDefinition>
9     <Topic>TOTAL_VALUE</Topic>
10    <UnderlyingTopic>/ett/order</UnderlyingTopic>
11    <Projection>
12      <Field>/109</Field>
13      <Field>SUM(/14 * /6) AS /71406</Field>
14    </Projection>
15    <Grouping>
16      <Field>/109</Field>
17    </Grouping>
18  </ViewDefinition>
19 </TopicMetaData>

```

Listing C.17: State of the World Example

C.3.10 Transaction Log

In order to support the high availability features of AMPS, a `TransactionLog` can be configured to keep a journal of messages published to an AMPS instance and all upstream replicas. The [High Availability](#) chapter covers the use cases

where a `TransactionLog` can be used to maximize the availability of your AMPS instance.

Table C.13: Transaction Log Configuration Parameters

Element	Description
JournalDirectory	Filesystem location where journal files will be stored.
PreAllocatedJournalFiles	The number of journal files AMPS will create as part of the server startup. Default: 2 Minimum: 1
MinJournalSize	The smallest possible journal size that AMPS will create. Default: 1 Minimum: 10M
BatchSize	A tuning parameter to configur the number of journal files to create at a time. When AMPS runs out of available journal files to write to, it will create <code>BatchSize</code> more journal files in a single pass. Default: 128 Maximum: 1024 Minimum: 1
Topic	If no topic is specified, AMPS will store all messages in the transaction log. If a <code>Topic</code> is specified, then
FlushInterval	The interval with which messages will be written to the journal file from AMPS. Default: 100ms Maximum: 100ms Minimum: 30us
O_DIRECT	Where supported, <code>O_DIRECT</code> will perform DMA directly from/to physical memory to a userspce buffer. Having this enabled can improve AMPS performance, however not all devices support <code>O_DIRECT</code> . Default: enabled

In the example listed in [Listing C.18](#) we demonstrate a transaction log where the journal file will be written to `./amps/journal`. When AMPS starts a single journal file will be pre allocated as noted by the `PreallocationJournalFiles` setting, and when a the first SOW is completely full, 128 new journal files will be created. This journal is only going to contain messages which match the the topic `orders` and also have a message type of `fix`. All messages that are going to be written to this file will be flushed in 40us intervals.

```

1 <AMPSConfig>
2   ...
3
4   <TransactionLog>
5     <JournalDirectory>./amps/journal/</JournalDirectory>

```

```

6 <PreallocatedJournalFiles>1</PreallocatedJournalFiles>
7 <MinJournalSize>10MB</MinJournalSize>
8 <BatchSize>128</BatchSize>
9 <Topic>
10 <Name>orders</Name>
11 <MessageType>fix</MessageType>
12 </Topic>
13 <FlushInterval>40ms</FlushInterval>
14 </TransactionLog>
15
16 ...
17 </AMPSSConfig>

```

Listing C.18: Transaction Log Configuration Example

C.3.11 SOW Statistics Interval

AMPS can publish SOW statistics for each SOW topic which has been configured. The `SOWStatsInterval` is specified as an interval (see [AMPS Configuration - Time Units](#)) between updates to the `/AMPS/SOWStats` topic.

Table C.14: SOW Statistics Interval Parameters

Element	Description
<code>SOWStatsInterval</code>	Interval for which SOW statistics are updated.

```

1 <AMPSSConfig>
2 ...
3 <SOWStatsInterval>10s</SOWStatsInterval>
4 ...
5 </AMPSSConfig>

```

Listing C.19: SOW Statistics Interval Example

Minidump Directory

The minidump directory is used to specify a location for AMPS to create a file which contains program information which is useful for support and diagnostics. AMPS will generate a minidump file on any crash event, or a minidump file can be generated at any point in time through the monitoring interface (see [Section D.2](#)).

Table C.15: Mini Dump Directory Parameters

Element	Description
MiniDumpDirectory	Location to store AMPS mini dumps. Default is /tmp.

```
1 <MiniDumpDirectory>/var/tmp</MiniDumpDirectory>
```

Listing C.20: Mini Dump Directory Example

Configuration Validation

Configuration validation can be used to enable or disable the validation checking performed by AMPS on the initialization of each instance. Disabling the configuration validation can cause AMPS to start in an invalid state or not properly log warnings or errors in the configuration file.



Configuration validation should only be used in testing or debugging. It is strongly not recommended to be used in a production or development environment.

Table C.16: Replication Destinations Parameters

Element	Description
ConfigValidation	Setting this to <code>disabled</code> will turn off AMPS configuration validation. The default is <code>enabled</code> which ensures that the current AMPS configuration meets valid parameter ranges and data types.

```
1 <AMPSConfig>
2   <ConfigValidation>enabled</ConfigValidation>
3 </AMPSConfig>
```

Listing C.21: Configuration Validation Example

Appendix **D**

Monitoring Interface Reference

D.1 Host Interface

The `host` URI contains information about the current operating system devices, such as the CPU, memory, disk and network. In addition, a host's network hostname and system timestamp time are also exposed through the monitoring interface.

CPUs

The `cpu` resource allows an administrator to view the CPU devices attached to the host. Selection of the `cpu` link in the `host` resource generates a list of all CPUs attached to the host, and also an aggregate `all` option.

Table D.1: CPU Statistics

Element	Description
<code>idle_percent</code>	Percent of CPU time that the system did not spend waiting on an I/O request to complete.
<code>iowait_percent</code>	Percent of CPU time spent waiting for I/O requests to complete.
<code>system_percent</code>	Percent of CPU utilization time which occurred while executing kernel processes.
<code>user_percent</code>	Percent of CPU utilization time which occurred while running at the application level.

Disks

The `disks` resource lists each of the disk devices attached to the host and permits the inspection of disk usage statistics. This information is a readily consumable version of the `/proc/diskstats` file. Statistics reported are based on the statistics monitoring update frequency (see `Interval` in [Section C.1.2](#)). This means, for example, that `reads` is the number of disk-reads for the given statistics update interval.

Table D.2: Disk Statistics

Element	Description
<code>in_progress</code>	Number of I/O requests waiting to be processed.
<code>read_time</code>	Number of milliseconds spent reading.
<code>reads</code>	Number of reads completed.
<code>reads_merged</code>	Reads and writes which are adjacent to each other may be merged for efficiency. This calculates the number of merged read/write operations.
<code>sectors_read</code>	Number of sectors read.
<code>sectors_written</code>	Number of sectors written.
<code>total_time</code>	Number of milliseconds spent performing I/O operations.
<code>weighted_cost</code>	Weighted cost is incremented at each I/O start, I/O completion, I/O merge or read of these stats by the number of I/Os in progress (<code>in_progress</code>) times the number of milliseconds spent doing I/O since the last update of this field. This can provide an easy measure of both I/O completion time and the backlog that may be accumulating.
<code>write_time</code>	Number of milliseconds spend writing.
<code>writes</code>	Number of writes performed.

Memory

The `memory` resource gives details about the system memory statistics. All statistics reported are based on the current system statistics reported by examining the `/proc/meminfo` file. These statistics are updated based on the statistics monitoring update frequency (see `Interval` in [Section C.3.2](#)). All memory statistics are reported in kB.

Table D.3: Memory Statistics

Element	Description
<code>available</code>	The total amount of memory available. Calculated as the sum of <code>free</code> , <code>buffers</code> and <code>cached</code> .

Continued on next page

TableD.3 -- continued from previous page

Element	Description
buffers	The amount of physical memory available for file buffers.
cached	The amount of physical memory used as cache memory.
free	The amount of physical memory left unused by the system.
in_use	The amount of memory currently in use. Calculated as <code>total - (free + buffers + cached)</code> .
swap_free	The amount of swap memory which is unused.
swap_total	The total amount of physical swap memory.
total	Total amount of RAM.

Name

The `name` resource displays the network DNS name for the host.

Network

The `network` resource allows an administrator to examine networking interface statistics on the host. Selecting the `network` resource displays a list of the network interfaces attached to the host. Selecting one of the interfaces will list the available properties. Rate based Statistics reported are based on the statistics monitoring update frequency (see `Interval` in [Section C.1.2](#)). For example, `collisions` for interface `eth0` would report the total number of collisions since the last statistics update interval.

Table D.4: Network Statistics

Element	Description
bytes_in	Number of bytes received by the interface.
bytes_out	Number of bytes transmitted by the interface.
carrier_errors	Total number of transmit and receive errors.
collisions	Number of collisions detected on the interface.
compressed_in	The number of compressed packets received by the interface.
compressed_out	The number of compressed packets sent by the interface.
drop_in	The number of receive packets dropped by the interface driver.
drop_out	The number of sent packets dropped by the interface driver.

Continued on next page

TableD.4 -- continued from previous page

Element	Description
errors_in	The total number of receive errors detected by the device.
errors_out	The total number of send errors detected by the device.
fifo_errors_in	The number of FIFO buffer errors while receiving.
fifo_errors_out	The number of FIFO buffer errors while sending.
frame_errors	The number of packet framing errors.
multicast	The number of multicast frames transmitted or received by the device driver.
packets_in	The total number of packets received by the interface.
packets_out	The total number of packets sent by the interface.

UTC time

The `utc_time` resource displays the system time on the host. Note: the `utc_time` time reflects the time on the host that the HTTP GET was processed. This differs from all other resources in the `host` interface as their update frequencies are determined by the `Interval` tag. For more information on the `Interval` tag see [Admin Parameters](#).

D.2 Instance Interface

The `Instance` resource provided by the AMPS monitoring interface is the administrative overview of a running AMPS instance. At a glance an administrator has access to a wide view of statistic and configuration information related to AMPS usage.

administrator

clients

Selecting the `clients` resource will list all connected clients by name. Selecting a single client will permit them to be disconnected.

authorization

Selecting the `authorization` resource will allow the `authentication` or `entitlement` resources to be reset. Selecting either one of these will present

a `reset` link which will call the reset function defined by the respective authentication or entitlement resource. For more information on building authentication and entitlements into AMPS, please refer to [Appendix E](#).

minidump

Selecting the `minidump` resource will create a minidump of the current running AMPS instance. The minidump will be saved in directory specified by the `MiniDumpDirectory`. See [Table C.15](#) for more information.

replication

Selecting the `replication` resource will list all currently configured replications. Selecting any individual replication destination will permit them to be downgraded.

clients

Selecting the `clients` resource will list all connected clients by name. Selecting a single client will grant the user the ability to view various properties regarding a client.

Table D.5: Client Statistics

Element	Description
<code>ack_count</code>	Number of acknowledgments sent.
<code>ack_count_per_sec</code>	Rate of acknowledgments sent.
<code>bytes_in</code>	Number of bytes received.
<code>bytes_in_per_sec</code>	Rate of bytes received.
<code>bytes_out</code>	Number of bytes sent.
<code>bytes_out_per_sec</code>	Rate of bytes sent.
<code>client_name</code>	Unique identifier of the client set during the <code>logon</code> .
<code>connect_time</code>	UTC time client connection is established.
<code>connection_name</code>	Name of the connection
<code>disconnect_count</code>	Number of times disconnected.
<code>is_connected</code>	Boolean value establishing
<code>messages_in</code>	Number of messages sent from client.
<code>messages_in_per_sec</code>	Rate of messages sent.
<code>publish_count</code>	Number of messages published.
<code>publish_count_per_sec</code>	Rate of messages published.
<code>queries_canceled</code>	Number of queries canceled by client.
<code>queries_canceled_per_sec</code>	Rate of queries canceled by client.

Continued on next page

TableD.5 -- continued from previous page

Element	Description
queries_completed	Number of queries completed.
queries_completed_per_sec	Rate of queries completed.
queries_requested	Number of queries.
queries_requested_per_sec	Rate of queries.
query_bytes_out	Number of query bytes sent.
query_bytes_out_per_sec	Rate of query bytes sent.
query_time	time taken for queries to complete.
queue_depth_out	Number of messages queued to be sent to client.
queue_max_latency	The age of the oldest item in the queue which has not yet been sent. This is used as a measure of how far behind AMPS believes a subscribing client is.
queued_bytes_out	Number of queued bytes waiting to be sent.
remote_address	IP and port of the client connection.
subscription_count	Number of subscriptions the client has requested.

config.xml

Selecting this will display the current AMPS configuration file, but default called `config.xml`.

config_path

Filesystem location of the configuration file.

cpu

The CPU resource lists properties related to overall CPU usage of the AMPS instance. Selecting the items below give more specific information to the type of CPU utilization being consumed by the AMPS user.

Table D.6: CPU Statistics

Element	Description
system_percent	Percent of CPU utilization time consumed while executing kernel processes.
total_percent	Total percent of CPU utilization.
user_percent	Percent of CPU utilization time consumed while processing non-I/O events.

cwd

The current working directory of where the AMPS instance was invoked from.

logging

The `logging` resource contains information about the resources consumed during various AMPS logging processes. Selecting a logging mechanism (console, file or syslog) will first list all logs of that particular type. Drilling down into one of those logs will pull up more granular information about logging. If a logging mechanism is not defined in the configuration, then the results will be blank when the logging resource is selected.

console

Below are the options available for reporting when `console` logging is enabled.

Table D.7: Console Logging Statistics

Element	Description
<code>bytes_written</code>	Number of bytes written to the console.
<code>exclude_errors</code>	Errors which are excluded from logging.
<code>include_errors</code>	Errors which are included during logging.
<code>log_levels</code>	Log level used to control logging output.
<code>target</code>	Console to which logging output is directed. Default is <code>stdout</code> .

file

Below are the options available for reporting when `file` logging is enabled.

Table D.8: File Logging Statistics

Element	Description
<code>bytes_written</code>	Number of bytes written to the console.
<code>exclude_errors</code>	Errors which are excluded from logging.
<code>file_name</code>	File defined in the <code>config.xml</code> where the log file is written to.
<code>file_name_mask</code>	Make of the logging output file name, if available.
<code>file_system_free_percent</code>	Amount of file system available.
<code>include_errors</code>	Errors which are included during logging.
<code>log_levels</code>	Log level used to control logging output.

Continued on next page

TableD.8 -- continued from previous page

Element	Description
rotation	Boolean representation denoting if log rotation is turned on
rotation_threshold	Log size at which log rotation will occur.

syslog

Below are the options available for reporting when `syslog` logging is enabled.

Table D.9: System Logging Statistics

Element	Description
bytes_written	Number of bytes written to the console.
exclude_errors	Errors which are excluded from logging.
facility	Integer enumeration of the logging facility used by syslog.
ident	Syslog name of the logging instance.
include_errors	Errors which are included during logging.
log_levels	Log level used to control logging output.
logopt	Bitfield of possible log options included. These values are configured in the config.xml file in the <code><Options></code> tag.

memory

AMPS can provide information regarding the process's memory usage in its RSS and VMSize via the `memory` resource in the monitoring interface.

Table D.10: AMPS Instance Memory

Element	Description
rss	The resident set size of the AMPS process.
vmsize	The virtual memory size of the AMPS process.

message_types

Information regarding the message types (or transports) used by AMPS are maintained in the `message_types` resource. AMPS can track the following information for all four of the message types supported: `fix`, `nvfix`, `soapfix` and `xml`.

Table D.11: AMPS Instance Message Types

Element	Description
module	The type of transport module defined.
name	The name of the transport type.
options	Any special options provided by the transport.
type	The transport type.

name

Name of the AMPS Instance.

pid

The process ID of the current ampServer process.

processors

Selecting the `processors` resource will list all the available message processors that the AMPS instance has invoked to handle messages. Each AMPS message processor will be listed individually, or selecting the `all` resource will list an aggregate of the available message processors. All AMPS message processors have the following attributes available:

Table D.12: AMPS Message Processors

Element	Description
bytes_published	Total number of bytes published.
bytes_published_per_sec	Rate at which bytes are published.
client_publishes	Number of publish messages from the client.
client_publishes_per_sec	Rate of publish messages from the client.
description	Descriptor of the processors resource.
last_active	Number of seconds since a processor was last active
matches_found	Number of messages found.
matches_found_per_sec	Rate of messages found.
messages_received	Number of messages received.
messages_received_per_se	Rate of messages received.

queries

The `queries` resource lists all available information regarding the query messages sent to AMPS.

queued queries

A count of all queries which have not yet completed processing.

replication

Selecting the `replication` resource will display a list of available downstream replication instances used by this instance of AMPS. Selecting an individual replication instance will display the following statistics.

Table D.13: Replication

Element	Description
<code>bytes_out</code>	number of bytes sent.
<code>bytes_out_per_sec</code>	rate of bytes sent.
<code>client_type</code>	specifies whether client is a replication source or destination.
<code>connect_time</code>	time connected to replication instance.
<code>disconnect_count</code>	number of times replication destination has been disconnected.
<code>disconnect_time</code>	total amount of time in which the replication destination has been disconnected.
<code>filter</code>	string used to select subset of topic records.
<code>is_connected</code>	Boolean telling whether replication destination is currently connected.
<code>message_type</code>	type of messages used to pass records to the client.
<code>messages_out</code>	number of messages sent.
<code>messages_out_per_sec</code>	rate of messages sent.
<code>name</code>	name of replication configuration
<code>pass_through</code>	Boolean stating whether messages can only be sent to one client, or can messages be sent on to other downstream clients.
<code>replication_type</code>	One of either <code>sync</code> or <code>async</code>
<code>topic</code>	The name of the topic which is being replicated.

SOW

Clicking the `sow` link will list all available `sow` topics for the AMPS instance. Selecting a single `sow` topic will list the following available statistics about the `sow` topic:

Table D.14: SOW interface

Element	Description
<code>file_system_free_percent</code>	Percent of file system disk space available.
<code>free_records</code>	Number of records available before another slab has to be allocated.
<code>mappings</code>	Number of unique records stored in the SOW
<code>max_datasize</code>	Largest message.
<code>max_records</code>	Maximum number of records which can be stored in a SOW topic.
<code>multirecords</code>	number of messages that span multiple records
<code>path</code>	File system location of the SOW topics file store.
<code>record_size</code>	Size of each message stored in the SOW topic.
<code>size</code>	Total size of the SOW topic file store. Can be computed by finding the product of <code>max_records</code> * <code>record_size</code> .
<code>slabs</code>	analogous to 'blocks' in memory management. This is the number of <code>mmaped</code> memory regions servicing the SOW topic.
<code>valid_keys</code>	number of distinct messages in the SOW - defined by the SOW topic key.

statistics

The `statistics` resource contains information regarding how AMPS monitors its own statistics.

Table D.15: Statistics Interface

Element	Description
<code>disk_per_sample</code>	
<code>file_name</code>	Location where statistics are stored. Default is <code>:memory:</code> which stores the statistics database in system memory.
<code>file_size</code>	Size on disk of the statistics database.
<code>interval</code>	Time in milliseconds between statistics database updates.

Continued on next page

TableD.15 -- continued from previous page

Element	Description
memory_used	Size in bytes of the system memory consumption of the statistics database.
queries	Number of queries processed from the statistics database.
time_per_sample	Time taken to process each statistics database query.
total_samples	Number of statistics database updates which have taken place since the AMPS server started.
total_time	Total amount of time spent publishing statistics

subscriptions

Each client which submits a `subscribe` command message is tracked by AMPS, and their relevant metrics are captured in the monitoring instance database. Selecting the `subscriptions` resource lists the available subscribers. Selecting a subscriber will list the available statistics below:

Table D.16: Subscriptions Interface

Element	Description
client_id	The ID of the subscribing client.
filter	Any filters applied to the subscribing topic.
is_bookmark	Boolean value to determine if the subscription is a bookmark.
is_oof_enabled	Boolean value to determine if the subscription has OOF (Out Of Focus Processing) enabled.
is_replication	Boolean value to determine if the subscription is applied to a replication.
message_type	Transport type of the subscription message. All return acknowledgments and messages use the same transport as the subscription.
send_empties	Boolean to determine if sending empty messages is required / permitted.
subscription_type	Type of subscription.
topic	Subscription topic.

uptime

The length of time that the AMPS instance has been running which conforms to a `hh:mm:ss.uuuuuu` format. This format is explained in [Table D.17](#)

Table D.17: Time formatting used in `uptime`

Abbreviation	Definition
hh	hours
mm	minutes
ss	seconds
uuuuuu	microseconds

user_id

The username for the owner for the `ampServer` process.

version

Version of the current running instance of AMPS.

views

Clicking the `views` link will give a list of the views defined in the configuration file for the AMPS instance. Clicking a view will display the detailed resources for views.

Table D.18: Subscriptions Interface

Element	Description
grouping	List of one or more fields which are used to determine message aggregation.
projection	The formula defined in the AMPS config for the computed transformation of one or more fields onto a new field.
queue_depth	The number of messages in the view which have not yet completed processing.
topic	The name of the new AMPS topic created by this view.
underlying_topic	The source topic used to compute the projected view.

Appendix **E**

Authentication and Entitlements

Within AMPS feature set is support for adding an authentication module, like Kerberos, to enable and require authentication prior to permitting a client to have access to an AMPS server. Additionally AMPS provides an entitlements module which allows an administrator to have control over the features which are granted access to an authenticated user. This chapter will cover how to configure AMPS to enable authentication, how to write an AMPS module to grant entitlements and how to logon/authenticate a client with an authentication-enabled AMPS instance.

E.1 Configuration

Configuration of the Authentication and Entitlements modules are covered in greater detail in greater detail in [Section C.3.3](#). A brief example of an Authentication module which is applied to an AMPS `Transport` is demonstrated below.

The first step in enabling Authentication for AMPS is to configure the `Module` which will be used throughout the configuration as the Authentication module. An example of this is listed below in [Listing E.1](#)

```
1 <AMPSConfig>
2   ...
3   <Modules>
4     <Module>
5       <Name>authentication1</Name>
6       <Library>libauthenticate_customer001.so</Library>
7       <Options>
8         <LogLevel>info</LogLevel>
```

```

9     <Mode>debugging</Mode>
10    </Options>
11    </Module>
12    ...
13    </Modules>
14    ...
15 </AMPSConfig>

```

Listing E.1: Global Configuration for Authentication

As shown in [Listing E.1](#), line 3 shows the `Modules` tag which is used to contain all of the `Module` definitions for the configuration file. A `Module` is a pluggable library which is defined in a `Module` tag and is then referenced by an AMPS feature which makes use of that `Module`. In this instance, our `Module` is named `authentication1` - as indicated on line 4 with the `Name` tag - and it references the `libauthenticate_customer001.so` shared library as the `Library` implementation, as defined on line 5. The `Options` are items which are passed into the `Module` and are handled internal to the library's implementation.

Now that we have defined and named our shared library `authentication1`, let us begin by implementing a `Transport` which uses this module for its Authentication. We will use [Listing E.2](#) as a simple example.

```

1 <AMPSConfig>
2   ...
3   <Transports>
4     <Transport>
5       <Name>fix-tcp-001</Name>
6       ...
7       <Authentication>
8         <Module>authentication1</Module>
9       </Authentication>
10    </Transport>
11    ...
12  </Transports>
13  ...
14 </AMPSConfig>

```

Listing E.2: Example of Transport with Authentication Enabled

The `Authentication` module used to define the `fix-tcp-001` transport is listed starting on line 7. The `Authentication` tag is used to define the supported type of module implemented, and the `Module` tag references the Name of the module listed in `auth:config-auth`. Our transport will now use require that users' credentials are authenticated via the `logon` command.

E.2 AMPS Administration

The AMPS Administration console provides a mechanism for the purpose of resetting the authorization state of currently logged in and authenticated clients

in AMPS.

During a reset, AMPS will attempt to create a new context for the respective module. On successful creation of a new context which will clear any cached state associated with the previous context, and then destroy the old context.

On an error, where the context could not be created AMPS will continue to operate with the old context until the problem can be corrected and a context reset can be successfully performed.

Greater detail in accessing the links via the AMPS administration console is covered in [Section D.2](#).

E.3 AMPS Guarantees

For AMPS to enable authentication and entitlement, there are first a few rules that an AMPS client must follow in order to guarantee a successful logon. They are as follows:

1. Send `logon` command with `UserId` and `Password` set in the header fields of the command. The `logon` must also request a `processed` acknowledgement which is used in returning the state of authentication.
2. Receive the processed acknowledgement.
3. If the `Status` of the processed acknowledgement is failure with a `Reason` of `Retry` goto step 4. Otherwise skip to step 5.
4. The `processed` ack will contain a `UserId` and `Password` to use in retrying the authentication. For most authentication mechanisms, this first retry will have a `Password` which contains a server side token (the server nonce) that is used to perform the second step of the client-side authentication and logon a second time. For other authentication mechanisms, the `UserId` can be modified to perform hygiene or aliasing, to support aliases or correction of common authentication errors (deployment specific). Using this new `UserId` and `Password` token, the client will re-authenticate on the client side and go to step 1.
5. If not failing because of the retry reason, then the reason must be invalid credentials and the client should notify the user and stop.

Other scenarios that a client developer will need to be aware of when working with an authenticated AMPS instance are listed below:

When the authentication module returns `AMPS_FAILURE` from `amps_authenticate`, AMPS will disconnect the session. Any command issued after the session has been disconnected will be silently discarded.

AMPS will force an implicit `logon` command for clients which have not issued one explicitly. The `UserId` and `Password` will be empty strings in this case. Handling this case will be implementation specific, and will be an

exercise left to the developer responsible for implementing the authentication and entitlement modules. This arrangement places the responsibility of determining which sessions are permitted to remain connected on the authentication module.

E.4 Authentication Interface

For the Authentication Interface, all `_init` functions have the `AMPS_CONFIG_PATH` environment variables available to them. The `AMPS_CONFIG_PATH` is an absolute path to the AMPS configuration file.

AMPS Authentication Module Init

The `amps_authentication_module_init` method is used to initialize the authentication module.

```
{int amps_authentication_module_init}(
    amps_module_options options,
    amps_module_logger message,
    amps_module_allocator allocator);
```

Parameters	<code>options</code>	Vector of <code>amps_option</code> structures(<code>{0,0}</code> sentinel).
	<code>logger</code>	Logging function to use for any messages.
	<code>allocator</code>	Allocator to use in allocating memory that AMPS needs to free.
Returns	<code>AMPS_SUCCESS</code>	The module was successfully initialized.
	<code>AMPS_FAILURE</code>	An error occurred (should be logged if appropriate).

AMPS Authentication Module Terminate

The `amps_authentication_module_terminate` method is used to terminate the authentication module.

```
int amps_authentication_module_terminate();
```

Parameters	None	
Returns	<code>AMPS_SUCCESS</code>	The module was successfully terminated.

	<code>AMPS_FAILURE</code>	An error occurred (should be logged if appropriate).
Notes	AMPS guarantees that no function within this module will be executed unless <code>amps_authentication_module_init</code> is invoked first.	

AMPS Authentication Create Context

The `amps_authentication_create_context` method initializes and returns an authentication context.

```
amps_authentication_context
amps_authentication_create_context ()
```

Parameters	None
Returns	NULL on error, otherwise a valid <code>amps_authentication_context</code> .

AMPS Authentication Destroy Context

The `amps_authentication_destroy_context` is called to destroy a specified authentication context.

```
int amps_authentication_destroy_context (
    amps_authentication_context context);
```

Parameters	<code>context</code>	The authentication context to destroy.
Returns	<code>AMPS_SUCCESS</code>	The context was successfully destroyed.
	<code>AMPS_FAILURE</code>	An error occurred (should be logged if appropriate).

AMPS Authenticate

The `amps_authenticate` method is used to authenticate a user's credentials against a specified authentication context.

```
int amps_authenticate(
    amps_authentication_context context
    const char* user,
    const char* passwd,
    char**      userOut,
    size_t*     userOutLength,
    char**      passwdOut,
    size_t*     passwdOutLength);
```

Parameters	<code>context</code>	The authentication context to use to authenticate against.
	<code>user</code>	User name to authenticate.
	<code>passwd</code>	Password token to use for authentication.
	<code>userOut</code>	Alternative user name to use.
	<code>userOutLength</code>	Alternative password token to use for authentication.
	<code>passwdOut</code>	Alternative password token to use for authentication.
	<code>passwdOutLength</code>	Alternative password token length.
Returns	<code>AMPS_SUCCESS</code>	User has been authenticated.
	<code>AMPS_RETRY</code>	Retry authentication with alternative user/password.
	<code>AMPS_FAILURE</code>	Failed to authenticate user.
Notes	In protocols which may require retrying authentication with an alternate user name or password token, the function must return <code>AMPS_RETRY</code> to force the retry. The function must provide the alternate user name and length in the <code>userOut</code> and <code>userOutLength</code> pointer, and the alternate password and length via the <code>passwdOut</code> and <code>passwdOutLength</code> pointers. <code>userOut</code> and <code>passwdOut</code> must be changed to point at strings allocated by this function. These strings must be allocated using the allocator passed into the <code>amps_entitlement_module_init</code> , and are automatically freed by the AMPS server when no longer needed.	

E.5 Entitlements Module interface

AMPS Entitlement Module Init

The `amps_entitlement_module_init` method is used to initialize the entitlement module.

```
int amps_entitlement_module_init(
    amps_module_options options,
    amps_module_logger logger,
    amps_module_allocator allocator);
```

Parameters	<code>options</code>	Vector of <code>amps_option</code> structures(<code>{0,0}</code> sentinel) (from config).
	<code>logger</code>	Logging function to use for any messages.
	<code>allocator</code>	Allocator to use in allocating memory that needs to be freed by AMPS.
Returns	<code>AMPS_SUCCESS</code>	The module was successfully initialized.
	<code>AMPS_FAILURE</code>	An error occurred (should be logged if appropriate).

```
int amps_entitlement_module_terminate()
```

Purpose	Terminates the entitlement module.	
Parameters	None.	
Returns	<code>AMPS_SUCCESS</code>	The module was successfully terminated.
	<code>AMPS_FAILURE</code>	An error occurred (should be logged if appropriate).
Notes	AMPS guarantees that no function within this module will be executed unless <code>amps_entitlement_module_init</code> is invoked first.	

AMPS Entitlement Create Context

The `amps_entitlement_create_context` method initializes and returns an entitlement context.

```
amps_entitlement_context
amps_entitlement_create_context();
```

Parameters	None.
Returns	NULL on error, otherwise a valid <code>amps_entitlement_context</code> .

AMPS Entitlement Destroy Context

The `amps_entitlement_destroy_context` method is used to destroy the entitlement context.

```
int amps_entitlement_destroy_context(
    amps_entitlement_context context);
```

Parameters	<code>context</code>	The entitlement context to destroy.
Returns	<code>AMPS_SUCCESS</code>	The context was successfully destroyed.
	<code>AMPS_FAILURE</code>	An error occurred (should be logged if appropriate).

E.5.1 AMPS Entitlement Check

The `amps_entitlement_check` method checks if a user is entitled to access a specific resource.

```
int amps_entitlement_check(
    amps_entitlement_context context,
    const char* user,
    amps_entitlement_resource_type resourceType,
    const char* resource,
    const char** filter,
    int entitlement);
```

Parameters	<code>context</code>	The entitlement context to use for the check.
	<code>user</code>	User identifier to check.
	<code>resourceType</code>	Resource type.
	<code>resource</code>	Resource identifier.
	<code>filter</code>	The content filter for the resource (Reserved for future use).
	<code>entitlement</code>	Set to either <code>AMPS_READ_ALLOWED</code> or <code>AMPS_WRITE_ALLOWED</code> .

Returns	<code>AMPS_SUCCESS</code>	The entitlement check was successful.
	<code>AMPS_FAILURE</code>	An error occurred (should be logged if appropriate).

E.6 Additional Notes

- AMPS will cache values/responses so redundant queries will be minimized.
- AMPS will not be re-entrant on a single context.
- Memory allocated via the `amps_allocator` function (the error strings) will be freed by AMPS.
- Any options found in the configuration element will be sent via the `options` vector. For example, `<SPN>foo.baml.com</SPN>` is included in the Authentication element, the vector would look like: `{{SPN, foo.baml.com}, {0,0}}`.
- The `Module` provided will use `dlopen/dlsym` to determine if it exports the correct symbols. If not, a critical error will be emitted and AMPS will shutdown. This allows a full path to a library to be provided or a simple library name to force using the `LD_LIBRARY_PATH`.
- A function to clear the authentication or entitlement state would clear cached data within AMPS and terminate and re-initialize the library.
- Using the same library for both authentication and entitlement will result in an error.

Appendix **F**

Glossary

acknowledgment a networking technique in which the receiver of a message is responsible for informing the sender that the message was received.

conflation the process of merging a group of messages into a single message. e.g. when sending acknowledgment messages for a group of sequential messages, sending only the most recent message can be used to conflate all messages which have outstanding acknowledgments waiting to be processed.

filter a text string that is used to match a subset of messages from a larger set of messages.

message expiration the process where the life span of records stored are allowed limited.

message type the data format used to encapsulate messages

oof (out of focus) the process of notifying subscribing clients that a message which was previously a result of a SOW or a SOW subscribe filter result has either expired, been deleted from the SOW or has been updated such that it no longer matches the filter criteria.

replica a downstream copy of records already stored in the SOW, Transaction Log or Replica of another instance of AMPS.

replication the process of duplicating the messages stored into an AMPS instance for the purpose of enabling high availability features.

replication source an instance of AMPS which is the primary recipient of a published message which are then sent out to a replication destination.

replication destination the recipient of replicated messages from the replication source.

slow client A client that is over-subscribed and being sent messages at a rate which is faster than it can consume.

SOW (State of the World) the last value cache used to store the current state of messages belonging to a topic.

topic a label which is affixed to every message by a publisher which used to aggregate and group messages.

transport the network protocol used to to transfer messages between AMPS subscribers, publishers and replicas.

transaction log a history of all messages published which can be used to recreate an up to date state of all messages processed.

view a data relation which is constructed from the records of a SOW topic.

List of Figures

3.1	Publish and Subscribe	11
3.2	Topic Based Pub/Sub	12
4.1	A SOW topic named <code>ORDERS</code> with a key definition of <code>/Key</code>	16
4.2	Updating the <code>MSFT</code> record by matching incoming message keys	16
5.1	SOW Query Sequence Diagram	21
5.2	SOW-And-Subscribe Query Sequence Diagram	22
8.1	TCP/IP packet	37
16.1	<code>sow_and_subscribe</code> example	72
16.2	<code>sow_and_subscribe</code> with <code>State</code> filter	73
16.3	<code>sow_and_subscribe</code> with <code>oof</code> enabled	74
16.4	OOF message	75
16.5	Initial <code>sow_and_delta_subscribe</code>	76
16.6	<code>delta_publish</code> message after a game	76
16.7	<code>publish</code> and <code>oof</code> after a trade	77
16.8	The final client table	78
20.1	Synchronous Persistence Acknowledgment	103
20.2	Asynchronous Persistence Acknowledgment	104
20.3	Persisted Ack strategy	105
20.4	Single AMPS instance	109
20.5	Simple High Availability Pair	110
20.6	Complex High Availability with Regional Replication	113
21.1	AMPS View Server Deployment Configuration	120
21.2	AMPS GUI Instance With <code>sow_and_subscribe</code>	121
21.3	AMPS Message Publish Update	122
21.4	AMPS OOF Processing	123

List of Tables

1.1	Documentation Conventions	3
1.2	Version Number Components	5
2.1	AMPS Distribution Directories	6
3.1	Topic Regular Expression Examples	13
4.1	Topic Definition Configuration Description	17
6.1	Escape Sequences	26
6.2	Logical AND with NULL/NaN Values	29
6.3	Logical OR with NULL/NaN Values.	29
7.1	Regular Expression Meta-characters	31
7.2	Regular Expression Repetition Constructs	31
7.3	Regular Expression Behavior Modifiers	32
8.1	TCP/IP Transport configuration parameters	35
10.1	Log Levels	44
10.2	Log filename masks	46
10.3	Log file rotation units	46
10.4	Logging options available for SYSLOG configuration.	49
10.5	Comparison of AMPS log severity to Syslog severity.	50
10.6	AMPS Error Categories.	51
11.1	<code>/AMPS/ClientStatus</code> Format Fields	54
11.2	Client Status FIX Format Fields	55
12.1	Acknowledgment messages supported by AMPS.	57
12.2	Commands and supported acknowledgment types.	58
13.1	Topic Replica Configuration Parameters	61
14.1	<code>ORDERS</code> table identifiers	63
14.2	Aggregate functions.	65

17.1 Parameters for <code>amps_sow_dump</code> .	79
17.2 Parameters for <code>amps_journal_dump</code> .	82
17.3 Parameters for <code>amps_err</code> .	82
18.1 Memory estimation equation.	87
18.2 Example memory estimation.	87
18.3 Minimum SOW size.	88
18.4 Maximum SOW size.	88
18.5 Maximum Message Size allowed in SOW.	88
18.6 Network capacity formula	89
18.7 Network capacity formula	90
A.1 FIX Header Fields - sorted by FIX Value	126
A.2 FIX Header Fields - sorted by Name	127
A.3 XML Header Fields - sorted by Name	128
A.4 Header Fields - sorted by Name	131
B.1 Header fields used in a <code>delta_publish</code>	132
B.2 Ack types supported by <code>delta_publish</code>	133
B.3 Header fields supported by <code>delta_subscribe</code> .	135
B.4 Ack types supported by <code>delta_subscribe</code>	136
B.5 Header fields supported by <code>logon</code> .	138
B.6 Ack types supported by <code>logon</code>	138
B.7 Header fields supported by <code>publish</code>	139
B.8 Ack types supported by <code>publish</code>	140
B.9 Header fields supported by <code>sow_and_delta_subscribe</code> .	142
B.10 ack types supported by <code>sow_and_delta_subscribe</code>	143
B.11 Header fields supported by <code>sow_and_subscribe</code> .	144
B.12 ack types supported by <code>sow_and_subscribe</code>	145
B.13 Options types supported by <code>subscribe</code>	145
B.14 Header fields supported by <code>sow_delete</code> .	147
B.15 ack types supported by <code>sow_delete</code>	148
B.16 Header fields supported by <code>sow</code> .	150
B.17 ack types supported by <code>sow</code>	150
B.18 Header fields supported by <code>start_timer</code> .	152
B.19 Header fields supported by <code>stop_timer</code> .	152
B.20 ack types supported by <code>sow</code>	152
B.21 Header fields supported by <code>subscribe</code> .	153
B.22 ack types supported by <code>subscribe</code>	154
B.23 Options types supported by <code>subscribe</code>	154
B.24 Header fields supported by <code>unsubscribe</code> .	156
B.25 ack types supported by <code>unsubscribe</code>	157
C.1 AMPS Configuration - Time Units	161
C.2 AMPS Configuration - Byte Units	162
C.3 AMPS Configuration - Numeric Units	162
C.4 Name Parameters	164
C.5 Admin Parameters	164
C.6 Module Parameters	165
C.7 Message Type Parameters	167

C.8 Transport Parameters	168
C.9 Logging Parameters	170
C.10 State of the World Parameters - TopicDefinition	171
C.11 State of the World Parameters - ReplicaDefinition	173
C.12 State of the World Parameters - ViewDefinition	174
C.13 Transaction Log Configuration Parameters	175
C.14 SOW Statistics Interval Parameters	176
C.15 Mini Dump Directory Parameters	177
C.16 Replication Destinations Parameters	177
D.1 CPU Statistics	178
D.2 Disk Statistics	179
D.3 Memory Statistics	179
D.4 Network Statistics	180
D.5 Client Statistics	182
D.6 CPU Statistics	183
D.7 Console Logging Statistics	184
D.8 File Logging Statistics	184
D.9 System Logging Statistics	185
D.10 AMPS Instance Memory	185
D.11 AMPS Instance Message Types	186
D.12 AMPS Message Processors	186
D.13 Replication	187
D.14 SOW interface	188
D.15 Statistics Interface	188
D.16 Subscriptions Interface	189
D.17 Time formatting used in <code>uptime</code>	190
D.18 Subscriptions Interface	190

Listings

7.1	Filter Regular Expression Example	30
7.2	Case Insensitive Regular Expression	31
7.3	Suffix Matching Regular Expression	31
7.4	Case Insensitive Prefix Regular Expression	31
7.5	Raw String Example	32
7.6	Regular String Example	32
7.7	Topic Regular Expression	33
8.1	TCP with FIX transport example.	34
8.2	TCP/IP configuration transport example	35
9.1	FIX message type configuration example.	38
9.2	Simple <code>publish</code> message in XML format.	40
16.1	Topic Configuration	70
18.1	Example of FIX Transport with Slow Client Configuration	94
20.1	Replication Source Example	116
20.2	Replication Destination Example	117
B.1	SOAP <code>publish</code> example	140
B.2	SOAP <code>sow_delete</code> example	149
C.1	Simple AMPS configuration file	158
C.2	File tokens used in configuration file	160
C.3	Date tokens used in configuration file	160
C.4	Expiration using seconds.	161
C.5	Expiration using seconds.	161
C.6	Environment variable used in configuration	162
C.7	AMPS_CONFIG_DIRECTORY Environment variable example	163
C.8	Name Example	164
C.9	Admin Example	164
C.10	Sample global config of authentication and entitlements modules.	165
C.11	Example of security enabled transports.	166
C.12	Message Types Example	168
C.13	Transports Example	169
C.14	Logging Example	171
C.15	SOW TopicMetaData Configuration	172
C.16	Replication example	173
C.17	State of the World Example	174

C.18 Transaction Log Configuration Example	175
C.19 SOW Statistics Interval Example	176
C.20 Mini Dump Directory Example	177
C.21 Configuration Validation Example	177
E.1 Global Configuration for Authentication	191
E.2 Example of Transport with Authentication Enabled	192

Index

Symbols

/AMPS/ClientStatus, 53

/AMPS/SOWStats, 54

AMPS

-- connectivity, 34

-- message protocols, 34

-- network protocols, 34

-- transports, 34

60East Technologies, 5

A

ack, 57

-- completed, 57

-- none, 57

-- persisted, 57

-- processed, 57

-- received, 57

-- stats, 57

AckType, 136, 140, 148

-- AckType, 134

-- completed, 57, 133, 135, 136, 142--
145, 147, 148, 150, 151, 153, 154,
157

-- none, 57, 133, 135, 136, 138, 139,
142--145, 147, 148, 150, 152--154,
156, 157

-- parsed, 153

-- persisted, 57, 102, 103, 111, 112,
115, 133, 136, 139, 140, 143, 145,
147, 148, 151, 153, 154, 157

-- processed, 57, 58, 111, 114, 133--
136, 138--140, 142--145, 147, 148,
150, 151, 153, 154, 156, 157

-- received, 57, 58, 133, 135, 136, 138,
139, 142--145, 147, 148, 150, 153,
154, 156, 157

-- stats, 57, 119, 124, 133, 135, 136,
142--148, 150, 151, 153, 157

Admin console

-- configuration, 164

Admin view, 8

aggregate functions, 65

-- null values, 65

aggregation, 65

AMPS

-- capacity, 86

-- commands, 132

-- demos, 7

-- events, 53

-- fix, 125

-- installation, 6

-- internal topics, 53

-- logging, 42

-- operation and deployment, 86

-- queries, 20

-- starting, 7

-- State, 15

-- topics, 12, 53

-- Utilities, 79

-- XML, 125

amps_journal_dump, 81

amps_sow_dump, 79

ampserr, 52

Authentication

-- configuration, 165

authentication, 181, 191

authorization, 191

avg, 65

B

backlog, 36

basics, 6

BatchSize, 22, 23, 150

Bookmark

-- epoch, 101

- zero, 101
- bookmarks, 104
- C**
- caching, 15
- capacity planning, 86
- client
 - backlog, 36
 - disconnect, 36
 - offlining, 36
 - slow, 36
 - status, 53
- client events, 53
- ClientStatus, 53
- Command, 146
- command, 132
 - Ack, 104
 - ClientStatus, 152
 - OOF, 74, 75, 146
 - ack, 57, 104--106, 133, 138, 152, 156
 - completed, 136
 - delta_publish message after a game, 76
 - delta_publish, 71, 76, 77, 132--135, 137
 - delta_subscribe, 135--137, 141
 - delta_subscription, 136
 - group_begin, 20, 22, 23, 121, 143, 145, 150
 - group_end, 20, 22, 23, 121, 150
 - logon, 102, 104, 111, 112, 137, 138, 192, 193
 - oof, 77, 122
 - publish, 39, 41, 77, 121, 137, 139--141
 - sow_and_delta_subscribe, 75--78, 123, 141--143
 - sow_and_subscribe, 21--24, 70, 72--75, 78, 120--124, 141, 144--146
 - sow_and_subscribe, 119
 - sow_delete, 146--148
 - sow, 20, 22--24, 33, 70, 141, 144, 147, 149--152
 - start_timer, 151, 152
 - stop_timer, 151--153
 - subscribe, 58, 101, 135, 144, 145, 153, 154, 156
 - subscription, 121, 156
 - unsubscribe, 156, 157
 - delta_publish, 58, 132
 - delta_subscribe, 58, 135
 - logon, 58
 - oof, 70
 - publish, 58, 139
 - sow, 59, 149
 - sow_and_delta_subscribe, 58, 141
 - sow_and_subscribe, 59, 144
 - sow_delete, 59, 146
 - start_timer, 151
 - stop_timer, 152
 - subscribe, 59, 153
 - unsubscribe, 59, 156
- commands
 - logon, 137
- Configuration, 158, 164
 - Admin console, 164
 - Authentication, 165
 - Entitlements, 165
 - Instance name, 164
 - Modules, 165
- configuration
 - AMPSConfig, 159
 - Admin, 159, 164
 - Destination, 173
 - Expiration, 161
 - InetAddr, 159, 173
 - Logging, 159
 - MessageType, 18, 173
 - Name, 164
 - Protocol, 159
 - Replication, 173
 - SOWStatsInterval, 54
 - TopicDefinition, 18, 56
- admin, 96
- mini dump, 176
- monitoring interface, 96
- SOWStatsInterval, 176
- TransactionLog, 174
- validation, 177
- view topic, 173
- conflation, 111
- connectivity, 34
- content filtering, 25
 - NaN, 28
 - NULL, 28
- count, 65
- D**
- delta

- publish, 132
- delta_publish, 132
- delta_subscribe, 135
- demo applications, 7
- deployment, 86
- disconnect, 36
- document conventions, 3

E

- engine
 - statistics, 54
- Entitlement
 - configuration, 165
- entitlement, 181
- entitlements, 191
- epoch bookmark, 101
- error categories, 50
- Errors
 - ampserr, 52
 - error categories, 50
- event topics, 53
- events, 53
- extracting records, 20

F

- falling behind, 36
- filters, 25
- FIX, 125
- fix, 34
- FIX messages, 9
- formatting of documentation, 3
- functions
 - aggregate, 65
 - -- null values, 65

G

- Glossary, 200

H

- High Availability, 101
 - bookmarks, 104
 - configuration, 115
 - deployment examples, 108
 - fail-over examples, 114
 - live subscription, 106
 - points of failure, 114
 - publish, 104
 - publisher responsibilities, guarantees, 111
 - regional replication, 112

- replication destination, 117
- replication source, 116
- subscribe, 106
- transaction log, 101
- high availability
 - replication, 102
- highlights, 1

I

- installation, 6
- internal event topics, 53
- intro to topics, 12
- IS NULL, 28

K

- kerberos, 191

L

- last value cache, 15
- Logging
 - configuration, 170
 - logging, 42, 184
 - file, 184
 - memory, 185
 - syslog, 185
 - logging:console, 184
 - logon, 137

M

- message expiration, 67
- Message Header
 - AckType, 133, 136, 138, 140, 143, 145--147, 152, 154
 - AckTyp, 57, 156
 - BatchSize, 144
 - BkMrk, 106
 - Bookmark, 101
 - ClientName, 138
 - CmdId, 146, 155
 - Matches, 136, 143, 145, 148
 - OOF, 74, 75
 - Options, 106, 144, 145, 154
 - Password, 193
 - Reason, 58
 - RecordsDeleted, 148
 - RecordsReturned, 136, 143, 145
 - SendOOF, 70, 72, 73, 75
 - SeqNo, 104--108
 - SowKeys, 148
 - SowKey, 148

- Status, 58, 133, 136, 143, 145, 146, 152
 - SubId, 58, 156
 - TopicMatches, 136, 143, 145, 148
 - UserId, 193
 - ackType, 133, 140, 148
 - bookmark, 104, 138
 - message protocols, 34
 - message types, 38
 - configuration, 167
 - fix, 38, 39
 - nvfix, 38
 - XML, 38, 40
 - MessageLength, 24
 - minidump, 95, 176, 182
 - Modules
 - configuration, 165
 - monitoring interface, 96, 97, 178
 - administration
 - replication, 182
 - configuration, 96
 - host, 97, 178
 - cpu, 178
 - disks, 179
 - memory, 179
 - name, 180
 - network, 180
 - utc_time, 181
 - instance, 97, 181
 - administrator, 181
 - clients, 182
 - config.xml, 183
 - config_path, 183
 - cpu, 183
 - cwd, 184
 - logging, 184
 - memory, 185
 - message_types, 185
 - name, 186
 - pid, 186
 - processors, 186
 - queries, 187
 - replication, 187
 - sow, 188
 - statistics, 188
 - subscriptions, 189
 - uptime, 189
 - user_id, 190
 - version, 190
 - views, 190
 - output formatting, 98
 - csv, 99
 - rnc, 100
 - xml, 98
 - time range selection, 97
- N**
- network protocols, 34
 - null values, 65
- O**
- offlining, 36
 - oof, 70
 - operating systems, 2
 - operation, 86
 - Operation and Deployment
 - minidump, 95
 - slow clients, 93
 - Options
 - live, 146, 155
 - no_empties, 146, 155
 - none, 145, 154
 - oof, 146, 155
 - replace, 146, 155
 - send_keys, 146, 155
 - organization, 2
 - Out of Focus, 70
 - overview, 1
- P**
- platforms, 2
 - playback, 101
 - pub/sub, 11
 - Publish
 - high availability, 104
 - publish, 11, 139
 - publish and subscribe, 11
- Q**
- query
 - filters, 25
 - QueryId, 20
- R**
- raw strings, 32
 - RecordSize, 17
 - Regular Expressions
 - raw strings, 32
 - regular expressions, 12
 - topics, 12
 - replay, 101

Replication, 173
replication, 101, 102, 187

S

slow client, 36
slow clients, 93
SOW, 15, 171
-- configuration, 16, 171
-- content filters, 25
-- queries, 16
-- query filters, 25
-- RecordSize, 17
-- statistics, 54
-- topic definition, 16
sow, 149
SOW events, 53
SOW Queries, 20
sow_and_delta.subscribe, 141
sow_and_subscribe, 144
sow_delete, 146
SowKey, 24, 137, 148, 149
SowKeys, 149
spark utility, 9
start_timer, 151
starting, 7
State of the World (SOW), 15
State of the World events, 53
statistics
-- SOW, 54
stop_timer, 152
storage, 15
Subscribe
-- high availability, 106
subscribe, 11, 153
sum, 65
support, 4
-- channels, 5
-- technical, 4
supported platforms, 2
syslog, 185

T

tag reference, 125
TCP/IP, 34
technical support, 4
Topic
-- replication, 173
topic
-- ClientStatus, 53
-- SOWStats, 54

topics
-- regular expressions, 12
transaction log, 101
transactions, 101
transports, 34, 168
troubleshooting
-- mini dump, 176

U

unsubscribe, 156
Utilities, 79
-- amps_journal_dump, 81
-- amps_sow_dump, 79
-- ampserr, 79

V

View Topic
-- configuration, 173

W

web console, 96, 178

X

XML, 34, 125