AMPS Configuration Reference Guide

AMPS Configuration Reference Guide

4.3

Publication date Oct 29, 2015

Table of Contents

Chapter 1. AMPS Configuration Reference Guide

This reference is targeted to those who have familiarized themselves with the *AMPS User Guide*, and are ready to dive into configuring some of the more advanced features of AMPS. This guide complements the *AMPS User Guide* by presenting the full set of options for the AMPS configuration file.

1.1. AMPS Configuration Basics

If you have not become familiar with the *AMPS User Guide*, in particular the *Getting Started* chapter, please start there before reading this guide.

The easiest way to create a custom XML configuration file for AMPS is to start with the sample configuration file produced by the `--sample-config` flag to AMPS. Example 1.1 shows a simplified sample configuration file.

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- Sample AMPS configuration

     This file defines an AMPS instance that provides publish and
     subscribe, topic filtering, and content filtering for JSON
 messages.
     The instance provides messaging services on port 9007 of the
 server.
     This configuration also provides an adminstrative interface on
     port 8085, and logs serious messages (error and higher severity)
 to
     stdout.

     This sample file does not configure State of the World (SOW)
 Topics,
     Transaction Logs, Aggregation and Views, Historical Query,
 Replication,
     Authentication and Entitlement, Conflating Topic Replicas, or
 other
     features of AMPS.

     More details for the featuers available and how to configure
 them are
     provided in the AMPS User Guide and the AMPS Configuration
 Reference.
     Both are available at http://crankuptheamps.com/documentation/

  -->
```

```
<AMPSConfig>

  <!-- Name of the AMPS instance -->

  <Name>AMPS-Sample</Name>

  <!-- Configure the administrative HTTP server on port 8085

       This HTTP server provides admin functions and statistics
       for the instance
   -->

  <Admin>
    <InetAddr>localhost:8085</InetAddr>
  </Admin>

  <!-- Configure a transport for JSON messages over TCP on port 9007
    -->

  <Transports>
    <Transport>
      <Name>json-tcp</Name>
      <Type>tcp</Type>
      <InetAddr>9007</InetAddr>
      <MessageType>json</MessageType>
      <Protocol>amps</Protocol>
    </Transport>
  </Transports>

 <!-- Log messages of severity 'error' and higher to stdout -->

  <Logging>
    <Target>
      <Protocol>stdout</Protocol>
      <Level>error</Level>
    </Target>
  </Logging>

</AMPSConfig>
```

Example 1.1. 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 the connection target for the administration port. 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 4.1.

Next we describe 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 this sample, we keep things simple by focusing on JSON messages over `tcp`. In AMPS, each key to defining each transport is to give them a unique `InetAddr` port and specify the type of message AMPS will process on that port using `MessageType` tag. The `MessageType` tells AMPS how to parse the incoming messages on a specific port. In the above example, messages are coming in on port 9007, and AMPS uses the JSON parser to parse the body of the message. AMPS also requires a `Protocol` tag for the `Transport`, which specifies the format of the commands to AMPS. In this case, we use the standard `amps` protocol. (Older AMPS applications and AMPS installations may require a different protocol format, such as `fix` or `xml`. There's no functional difference between these protocols, but the AMPS server and the clients need to use the same protocol format to successfully exchange messages.)

The last portion of the configuration is Logging. In the above example, the `Logging` tag defines only one log target, but it's quite common to have one or more `Logging` targets. Again referring to the example, all logging messages that are at `error` level and above will be logged to the logging `Protocol` of `stdout`. 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 in the *AMPS User Guide* and in Chapter 8 of this reference.

AMPS Configuration File Special Characters

In AMPS there are a few special characters that you should be aware of when creating your configuration file. These characters can provide some handy short cuts and make configuration creation easier, but you should also be aware of them so as not to introduce errors.

State of the World File Name

When specifying the file for a State of the World database, using the `%n` string in the log file name specifies that the AMPS server will use the message type and topic name in that position to create a unique filename. Example 1.2 shows how to use this in the AMPS configuration file.

```
 <SOW>
    <TopicDefinition>
      <Topic>Customers</Topic>
      <FileName>./sow/%n.sow</FileName>
      <MessageType>json</MessageType>
      <Key>/customerId</Key>
    </TopicDefinition>
</SOW>
```

Example 1.2. SOW file name tokens used in configuration file

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. Example 1.3 shows a file name token replacement in the AMPS configuration file.

```
<Logging>
  <Target>
    <Protocol>file</Protocol>
    <Level>info</Level>
    <FileName>log/log-%n.log</FileName>
    <RotationThreshold>2G</RotationThreshold>
  </Target>
</Logging>
```

Example 1.3. Log file name 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 Example 1.4.

```
<Logging>
  <Target>
    <Protocol>file</Protocol>
    <Level>info</Level>
    <FileName>
      log/log-\%Y-\%m-\%dT\%H\%m\%s.log
    </FileName>
    <RotationThreshold>2G</RotationThreshold>
  </Target>
</Logging>
```

Example 1.4. 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.

AMPS provides full support for the date tokens provided by the standard strftime function, with the exception of `%n`, as described above. The following table shows some of the most commonly used tokens:

Table 1.1. Commonly Used Date and Time Tokens

Token	Provides	Example
%a	Short weekday name	Fri
%A	Full weekday name	Friday
%b	Short month name	Feb
%B	Full month name	February
%c	Simple date and time	Fri Feb 14 17:25:00 2014
%C	Century	20
%d	Day of the month (leading zero if necessary)	05
%D	Short date format (MM/DD/YY)	02/20/14
%e	Day of the month (leading space if necessary)	5
%F	Short date format (YYYY-MM-DD)	2014-02-20
%H	Hour (00-23)	17
%I	Hour (00-12)	05
%j	Day of the year (001-366)	051
%m	Month (01-12)	02
%p	AM or PM	PM
%r	Current time, 12 hour format	05:25:00 pm
%R	Current time, 24 hour format	17:25
%T	ISO 8601 Time format	17:25:00
%u	ISO 8601 day of the week (1-7, Monday = 1)	5
%V	ISO 8601 week number (00-53)	07
%y	Year, last two digits	14
%Y	Year, four digits	2014
%Z	Timezone name or abbreviation (blank if undetermined)	PST

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, then 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).

```
<SOW>
  <TopicDefinition>
    ...
    <Expiration>86400s </Expiration>
    ...
  </TopicDefinition>
</SOW>
```

Example 1.5. Expiration Using Seconds

An even easier way to specify an expiration of one day is to use the following `Expiration`:

```
<SOW>
  <TopicDefinition>
     ...
     <Expiration>1d</Expiration>
     ...
  </TopicDefinition>
</SOW>
```

Example 1.6. Expiration Using Days

Table 1.2 shows a listing of the time units AMPS supports in the configuration file.

Table 1.2. 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. Table 1.3 shows a listing of the byte units AMPS supports in the configuration file.

Table 1.3. 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 1.4 contains a list of the exponents supported.

Table 1.4. AMPS Configuration - Numeric Units

Units	Description
k	10^3 - thousand
M	10^6 - million

To make it easier for users to remember the units, AMPS interval and byte units are not case sensitive.

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, then 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 wrapped in `${}` with the environment variable defined in the current user operating system environment. Example 1.7 demonstrates how the environment variable `ENV_LOG` is used to define a global environment variable for the location of the host logging.

```
<Logging>
  <Target>
    <Protocol>file</Protocol>
    <FileName>${ENV_LOG}</FileName>
    <Level>info</Level>
    <RotationThreshold>2G</RotationThreshold>
  </Target>
</Logging>
```

Example 1.7. 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, assume that AMPS was started with the following command at the command prompt:

```
%>./ampServer ../amps/config/config.xml
```

Given this command, the log file configuration option shown in Example 1.8 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`.

```
<Logging>
  <Target>
    <Protocol>file</Protocol>
    <FileName>
```

```
      ${AMPS_CONFIG_DIRECTORY}/logs/infoLog.log
    </FileName>
    <Level>info</Level>
    <RotationThreshold>2G</RotationThreshold>
  </Target>
</Logging>
```

Example 1.8. 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.

Chapter 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 *Getting Started* chapter of the *AMPS User Guide*, 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`. The generated file is a bare-bones configuration that allows AMPS to start, process JSON messages, and provide monitoring through the admin interface.

```
ampServer --sample-config > config.xml
```

The AMPS server also provides the ability to perform basic validation of the config file, using the `--verify-config` flag.

```
ampServer --verify-config config.xml
```

The validation process checks for errors in the configuration that would prevent AMPS from starting, and reports warnings and informational messages about the configuration file. However, the validation process does not ensure that the configuration file provided is suitable for any particular purpose.

Chapter 3. Instance Level Configuration

This chapter describes elements of the AMPS configuration that set parameters for the instance as a whole.

Table 3.1. Instance Level Configuration 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`.
`Group`	Identifies the replication group for this instance. If no `Group` element is present, the replication group for this instance is set to the `Name` of the instance. Set the group parameter when being able to refer to a set of instances makes your replication configuration simpler.
`RegexTopicSupport`	Sets whether this instance supports regular expression topic matching. When this option is `true`, clients can register subscriptions using regular expressions and receive messages for all matching topics. When this option is `false`, regular expression characters are interpreted as literal characters. Defaults to `true`.
`Authentication`	Sets the default authentication module to use for transports that do not explicitly specify an authentication module. Authentication modules verify the identity of a connected user. The module specified must be one of the modules configured in the `Modules` element or one of the authentication modules that AMPS loads by default. See Table 12.2 for the list of default modules. Defaults to `amps-default-authentication-module`.
`Entitlement`	Sets the default entitlement module to use for transports that do not explicitly specify an entitlement module. Entitlement modules enforce permissions for a connected user. The module specified must be one of the modules configured in the `Modules` element or one of the modules that AMPS loads by default. See for Table 13.2 for the list of default modules. Defaults to: `amps-default-entitlement-module`
`Authenticator`	Sets the default authenticator module to use for outgoing connections from AMPS that do not explicitly specify an authenticator module. Authenticator modules provide credentials to use for outgoing connections. The module specified must be one of the modules configured in the `Modules` element or one of hte modules that AMPS loads by default. Defaults to: `amps-default-authenticator-module`

```
<AMPSConfig>
    ....
    <Name>AMPS</Name>
    <Group>Sample-AMPS</Group>
    ....
</AMPSConfig>
```

Example 3.1. Instance-Level Configuration Example

3.1. SOW Statistics Interval

AMPS can publish SOW statistics for each SOW topic which has been configured. The `SOWStatsInterval` is specified as an interval (see Table 1.2) between updates to the `/AMPS/SOWStats` topic.

Table 3.2. SOW Statistics Interval Parameters

Element	Description
`SOWStatsInterval`	Interval for which SOW statistics are updated.

```
<AMPSConfig>
  ...
  <SOWStatsInterval>10s</SOWStatsInterval>
  ...
</AMPSConfig>
```

Example 3.2. SOW Statistics Interval Example

3.2. Minidump Directory

The minidump directory is used to specify a location for AMPS to create a file that 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 the *AMPS Monitoring Reference Guide*).

Table 3.3. Mini Dump Directory Parameters

Element	Description
`MiniDumpDirectory`	Location to store AMPS mini dumps. Default is `/tmp`. If the directory does not exist, AMPS creates the directory. The special value `disabled` configures AMPS not to produce mini dumps.

```
<MiniDumpDirectory>/var/tmp</MiniDumpDirectory>
```

```

```

Example 3.3. Mini Dump Directory Example

3.3. 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 disabled during testing or debugging. We strongly recommend against disabling configuration validation in a production or development environment.

Table 3.4. Config Validation Parameters

Element	Description
`ConfigValidation`	Setting this to `disabled` will turn off AMPS configuration validation. The default is `enabled`, ensuring that the current AMPS configuration meets valid parameter ranges and data types.

```
<AMPSConfig>
  <ConfigValidation>enabled</ConfigValidation>
</AMPSConfig>
```

Example 3.4. Configuration Validation Example

3.4. Tuning

The `Tuning` section of the configuration file sets instance-level parameters for tuning the performance of AMPS. In many cases, AMPS self-tunes to take advantage of the hardware and environment. However, explicitly setting tuning parameters is sometimes necessary in cases where an AMPS instance cannot determine the best value. For example, if multiple AMPS servers are running on the same system, 60East recommends disabling NUMA.

Use the `Tuning` element with care. Options in the `Tuning` element can affect AMPS performance, and the behavior of `Tuning` options may be version-specific.

Table 3.5. Tuning Parameters

Element	Description
`NUMA/Enabled`	Setting this to `disabled` will turn off AMPS NUMA tuning. The default is `enabled`, which affinitizes certain AMPS threads to specific processors.

Element	Description
	The default value of `enabled` produces significantly better performance when a single instance of AMPS is running on a given system. However, if multiple instances of AMPS are running on the same system, setting this value to `disabled` for all of the instances on the system may reduce contention among the instances and produce better overall performance.

```
<AMPSConfig>
    <Tuning>
       <NUMA>
          <Enabled>enabled</Enabled>
       </NUMA>
    </Tuning>
</AMPSConfig>
```

Example 3.5. Tuning Example

Chapter 4. Admin Server

The `Admin` tag is used to control the behavior of the administration server.

Table 4.1. Admin Parameters

Element	Description
`InetAddr`	Defines a port for the embedded HTTP admin server, which can then be accessed via a browser. This element can also specify an IP address, in which case the HTTP server listens only on that address. If no IP address is specified, the HTTP server listens on all available addresses.
`FileName`	Location for storing the statistics information reported by the Admin Server. default: `:memory:`
`Interval`	The refresh interval for the Admin Server to update gathered statistics. default: `10s` minimum: `1s`
`Authentication`	The authentication to use for the Administrative interface. This is an `Authentication` element, as described in Chapter 12.
`Entitlement`	The entitlement to use for the Administrative interface. This is an Entitlement element, as described in Chapter 13.

```
<Admin>
  <InetAddr>localhost:9090</InetAddr>
  <FileName>stats.db</FileName>
  <Interval>20s</Interval>
</Admin>
```

Example 4.1. Admin Example

Chapter 5. Modules

The `Modules` section of the AMPS configuration file is used to load, configure and define any plug-in modules used for this installation of AMPS. AMPS supports a wide variety of plug-in modules, as described in the *Extending AMPS Guide*.

The following steps are required to use a plug-in module:

1. Load the module and declare the name of the module.

2. Define the AMPS object that the module contains and give the object a name and pass any required options.

3. Use the module in a specific context.

For many modules, such as `Authentication` and `Entitlement` modules, steps 2 and 3 are performed at the same time. Steps 2 and 3 above are separate when a module must have the same definition across mutliple contexts (for example, a `MessageType` which may be used in a Transport, a SOW, a View, and replicated to other instances).

The available features of a `Module` are listed in Table 5.1.

Table 5.1. Module Parameters

Element	Description
`Name`	A plain text name for the module. This will be used as a reference when the module is used elsewhere in the AMPS configuration, and is also the name that AMPS will use for logging messages related to the module.
`Library`	The shared object file that contains the compiled module. This must contain the path, relative to the AMPS server's directory.
`Options`	A list of supported features for the implemented library. AMPS allows you to pass options to the module by specifying elements within the `Options` element. The exact options that the module requires, if any, are determined by the creator of the module.

Example 5.1 provides an example of an AMPS configuration using an authorization and entitlement plug-in module. 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.

```
<AMPSConfig>
...
  <Modules>
    <Module>
```

```
      <Name>authentication1</Name>
      <Library>libauthenticate_customer001.so</Library>
      <Options>
        <LogLevel>info</LogLevel>
        <Mode>debugging</Mode>
      </Options>
    </Module>
    <Module>
      <Name>entitlement1</Name>
      <Library>libentitlement_customer001.so</Library>
      <Options>
        <LogLevel>error</LogLevel>
        <Mode>prod</Mode>
      </Options>
    </Module>
    ...
  </Modules>
...
</AMPSConfig>
```

Example 5.1. 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 create one type of `Authentication` module for the instance as a whole, and then create instances of a different type of Authentication and Entitlement modules for each `Transport`, to ensure that our `Transports` are properly enabling authentication and entitlements. In this example, the `Authentication` and `Entitlement` modules configured for an individual `Transport` are used for that transport, and the instance level modules are used as a default for transports that do not specify any `Authentication` or `Entitlement`.

This is accomplished via an entry similar to Example 5.2.

```

<AMPSConfig>
...
  <Authentication>
    <Module>amps-no-authorization</Module>
  </Authentication>
  <Entitlement>
    <Module>amps-no-authorization</Module>
  </Entitlement>
...
  <Transports>
    <Transport>
      <Name>fix-tcp-001</Name>
...
      <Authentication>
        <Module>authenticate_customer001</Module>
      </Authentication>
      <Entitlement>
        <Module>entitlement_customer001</Module>
      </Entitlement>
    </Transport>
    <Transport>
      <Name>fix-tcp-007</Name>
      ...
      <Authentication>
        <Module>authenticate_customer007</Module>
      </Authentication>
      <Entitlement>
        <Module>entitlement_customer007</Module>
      </Entitlement>
    </Transport>

    <Transport>
      <Name>json-tcp<Name>
      <!-- does not specify Authentication or
           entitlement, uses instance-level
           modules -->
      ...
    </Transport>

  </Transports>
...
</AMPSConfig>
```

Example 5.2. Example of security enabled transports

Example 5.2 shows how 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 Example 5.1. Similarly, the `fix-tcp-007` transport is secured with the `authenticate_customer007` authentication module and the `entitlement_customer007` entitlement module. In contrast, the `json-tcp` transport does not define modules, and instead uses the authentication and entitlement modules specified at the instance level.

Chapter 6. Message Types

This tag defines the message types supported by the AMPS instance. A single AMPS instance can support multiple message types, as `MessageTypes` can contain multiple `MessageType` definitions.

`MessageType` definitions for `fix`, `nvfix`, `xml`, `json`, `bson`, and `binary` are automatically loaded by AMPS. You only need to define a new `MessageType` these if the settings for the message type need to be changed (for example, to create a custom FIX-based type that changes the `FieldSeparator` of the message).

AMPS also supports the ability to create a composite message type by combining a number of existing message types. Composite message types are defined using the `MessageType` configuration element.

Table 6.1. Message Type Parameters

Name	*Description*
`Name`	This element defines the name for the message type. The name is used to specify `MessageType` in other sections such as `Transport` and `TopicDefinition`. By default, AMPS loads message types for `fix`, `nvfix`, `soapfix`, `json`, `bson`, `xml` and `binary`.
`Module`	The element specifies the name of the module that will be loaded for this message type. By default, AMPS loads the modules that implement the following message types: `fix`, `nvfix`, `soapfix`, `json`, `bson` `xml` and `binary`. AMPS supports creating composite message types out of existing message types using the `composite-global` and `composite-local` modules, which are loaded by default.
`AMPSVersionCompliance`	Sets the version compatibility for the messages that AMPS sends to the `/AMPS/SOWStats` topic. When set to `2`, AMPS creates messages that use the field tags used by AMPS 2.X versions, which differ from the tags in the current version of AMPS. By default, this value is unset, and AMPS uses the field tags for the current version. 60East recommends leaving this value unset unless your application requires version 2.0 messages.
`Options`	Options to pass to a custom message type module. AMPS does not specify the format or type of the elements within an `Options` element. AMPS simply parses the XML and then sends the XML to the module. If you are configuring a custom message type, see the documentation for that message type module for details.

Name	*Description*
`FieldSeparator`	*Option*: Applies to `fix` and `nvfix` message types. Sequence of characters used to separate field items in a FIX message. Note: this field is the ASCII value of the char sequence.
`HeaderSeparator`	*Option*: Applies to `fix` and `nvfix` message types. 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.
`MessageSeparator`	*Option*: Applies to `fix` and `nvfix` message types. 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.
`EarlyTerminationOpti-mization`	*Option*: Applies to the `json` message type. By default, AMPS includes a optimization to allow the server to to only partially parse JSON messages. This may result in unexpected behavior for some messages. For example, given a message such as `{ "code" : 1, "data" : "some data", "code" : 2 }`, AMPS will report the value of `code` as `1` when this optimization is active. To ensure consistent results, in this mode AMPS always reports the first value for a field even when AMPS fully parses the message. When set to `false`, the optimization is disabled. AMPS will fully parse all JSON messages and report the last value for a field. For the message above, AMPS would report the value of `code` as `2`. Default: `true`
`Type`	*Obsolete* No longer used in AMPS 4.0 and later versions: to define a base message type to customize, use `Module`.
`MessageType`	*Required*: Applies to message types that use the `composite-lo-cal` or `composite-global` modules. For composite message types, the `MessageType` definition must contain one or more message type declarations that specify the types that the composite message type contains. See the *AMPS User Guide* for more information on composite mes-sage types.

```
<MessageTypes>
  <!-- Define a FIX-based message type with custom separators -->
  <MessageType>
    <Name>fix-custom</Name>
    <Module>fix</Module>
    <!-- The following are FIX specific options -->
    <FieldSeparator>1</FieldSeparator>
    <HeaderSeparator>2</HeaderSeparator>
    <MessageSeparator>5</MessageSeparator>
  </MessageType>

  <!-- Define a message type for a custom
       payload. 'type-module' must be the
       Name of a Module specified in the
       configuration. -->
  <MessageType>
      <Name>custom-payload</Name>
      <Module>type-module</Module>
  </MessageType>

  <!-- Define a composite message type
       that combines a json message and
       a custom-payload message. -->

  <MessageType>
      <Name>custom-composite</Name>
      <Module>composite-local</Module>
      <MessageType>json</MessageType>
      <MessageType>custom-payload</MessageType>
  </MessageType>

</MessageTypes>
```

Example 6.1. Message Types Example

Chapter 7. Transports

The `Transports` element configures how AMPS communicates with publishers and subscribers, as well as how AMPS accepts connections for replication. The Transports element is a container for one or more `Transport` elements. Each `Transport` is a combination of a network transport, an AMPS header protocol, and a message type.

A `Transport` also specifies the `Authentication` used to validate the users that connect, and the `Entitlement` used to enforce permissions for users that connect over that transport.

AMPS supports a variety of network transports, header protocols and message formats for communication between publishers and subscribers. This section describes how to configure a `Transport`.

Table 7.1. Transport Parameters

Element	Description
`Name`	The name to use for this Transport. This name appears in the AMPS log for messages related to the transport.
`InetAddr`	The port on which AMPS will listen for this transport. This element can also specify an IP address, in which case AMPS listens only on that address. If no IP address is specified, AMPS listens on all available addresses.
`Protocol`	This element defines the protocol to use for sending and receiving messages. The protocol is typically `amps`, the name of a specific protocol for interoperability with another system or a legacy application, or the name of a custom protocol module specified in the `Modules` element.

AMPS provides support for the following protocols:

Table 7.2. Protocols

Protocol Name	Description
`amps`	Standard AMPS messaging, using compact headers in JSON format. AMPS accepts `json` as a synonym for `amps` in a protocol declaration.
`fix-session`	FIX session protocol, for use with systems that publish FIX messages using this format.
`websocket-json`	Websocket protocol, using JSON format headers.
Legacy protocols	
`fix`	Standard AMPS messaging, using headers in FIX format.
`nvfix`	Standard AMPS messaging, using headers in NVFIX format.

Element	Description
	<table><tr><td>Protocol Name</td><td>Description</td></tr><tr><td>`soap`</td><td>Standard AMPS messaging, using headers in SOAP format.</td></tr><tr><td>`xml`</td><td>Standard AMPS messaging, using headers in XML format.</td></tr></table> 60East recommends using the `amps` protocol for general purpose AMPS messaging. When your application uses the the FIX session layer or Websockets, use those protocols. Older versions of AMPS used message headers in the same format as the message type: if your instance supports applications that expect to use a specific message type protocol, use that protocol in your `Transport` configuration.
`Type`	The type of Transport. Valid values include: `tcp`, `amps-replication`
`MessageType`	Defines the message type for this transport, and is a reference to the name of a specific message type defined in the `MessageTypes` section or one of the message types that AMPS loads by default. In this release, AMPS loads the following message types by default: `fix`, `nvfix`, `xml`, `json`, `bson` and `binary`.
`ReuseAddr`	Permits an AMPS instance to use a socket that is in a WAIT state. This can occur when AMPS has been restarted using the same `InetAddr` and the previous instance did not fully close the port. Valid values: `true` or `false` Default: `false`
`SlowClientDisconnect`	Define whether or not to disconnect clients based on the number of messages that are offlined for the client. This setting does not affect disconnection based on the `ClientMaxBufferThreshold`. AMPS always disconnects clients that exceed that threshold. Valid values: `enabled` or `disabled` (also accepts `true` or `false`) Default: `enabled` This option is not used when the transport type is `amps-replication`.
`ClientOffline`	Defines whether or not AMPS should offline messages a slow client to reduce memory pressure on the AMPS instnace when a client falls behind.

Element	Description
	Valid values: `enabled` or `disabled` (also accepts `true` or `false`) Default: `enabled` This option is not used when the transport type is `amps-replication`
`ClientBufferThreshold`	Defines how much memory a slow client can use before off lining. Units: Bytes Default: 50MB This option is not used when the transport type is `amps-replication`
`ClientMaxBufferThreshold`	The maximum amount of space, in bytes, that AMPS will use to buffer messages for a slow client. If a client exceeds this threshold, AMPS disconnects the client. This limit applies regardless of other slow client settings. Notice that when offlining is enabled, clients can also be disconnected based on the count of messages offlined (as set by the `ClientOfflineThreshold`). Units: Bytes Default: 1GB This option is not used when the transport type is `amps-replication`
`ClientOfflineThreshold`	Defines how many messages a slow client can offline before being disconnected. Units: Messages Default: 100K This option is not used when the transport type is `amps-replication`
`ClientOfflineDirectory`	Location to persist messages for a slow client in anticipation of it resuming message processing. Required if `ClientOffline` is enabled. Default: `/var/tmp` This option is not used when the transport type is `amps-replication`
`Entitlement`	Specifies the entitlement module to use for this transport. If no entitlement module is provided, the transport uses the default entitlement module for the instance. This element must contain a `Module` element with the `Name` of an entitlement module. If the module requires options, those options are provided in an `Options` element within the `Entitlement` element.

Element	Description
	Default: The module specified in the `Entitlement` for the instance (defaults to `amps-default-entitlement-module` if not provided)
`Authentication`	Specifies the authentication module to use for this transport. If no authentication module is provided, the transport uses the authentication module for the instance. This element must contain a `Module` element with the `Name` of an authentication module. If the module requires options, those options are provided in an `Options` element within the `Authentication` element. Default: The module specified in the `Authentication` element for the instance (defaults to `amps-default-authentication-module` if not provided)

```
<Transports>

  <!-- fix messages using TCP -->
  <Transport>
    <Name>fix-tcp</Name>
    <Type>tcp</Type>
    <InetAddr>9004</InetAddr>
    <ReuseAddr>true</ReuseAddr>
    <MessageType>fix</MessageType>
    <ClientBufferThreshold>
      4194304
    </ClientBufferThreshold>
    <ClientOffline>enabled</ClientOffline>
    <ClientOfflineThreshold>
      10000
    </ClientOfflineThreshold>
    <ClientOfflineDirectory>
      /var/tmp
    </ClientOfflineDirectory>
    <SlowClientDisconnect>true</SlowClientDisconnect>
  </Transport>

  <!-- nvfix messages using TCP -->
  <Transport>
    <Name>nvfix-tcp</Name>
    <Type>tcp</Type>
    <InetAddr>9005</InetAddr>
    <ReuseAddr>true</ReuseAddr>
    <MessageType>nvfix</MessageType>
  </Transport>

  <!-- xml messages using TCP -->
  <Transport>
    <Name>soap-tcp</Name>
    <Type>tcp</Type>
    <InetAddr>9006</InetAddr>
    <ReuseAddr>true</ReuseAddr>
    <MessageType>xml</MessageType>
  </Transport>

</Transports>
```

Example 7.1. Transports Example

Chapter 8. Logging

AMPS supports several different types of log formats, and multiple targets can be defined simultaneously.

Table 8.1. 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 1.3 for details on specifying file size.
`Level`	Defines a lower bound (inclusive) log level for logging. 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. This element can be used with the `Level` element. In that case, the AMPS will log all messages at `Level` and above, and in addition, will log errors at the levels specified by `Levels`. Valid values: `none`, `trace`, `debug`, `stats`, `info`, `warning`, `error`, `critical`, `emergency`
`IncludeErrors`	Additional errors that should be included when logging. If an error appears in this element, it will be logged regardless of the level of the error.
`ExcludeErrors`	Errors that should be excluded when logging. If an error appears in this element, it will not be logged regardless of the level of the error. If the same error appears in both `IncludeErrors` and `ExcludeErrors`, `ExcludeErrors` takes precedence and the error will not be logged.
`Ident`	Syslog identifier for the AMPS instance. Default: AMPS Instance Name
`Options`	A comma separated list of syslog options. If using `syslog`, 60East recommends using `LOG_CONS`, `LOG_NDELAY`, and `LOG_PID`. AMPS uses the standard options to syslog, as described in the syslog man page.

Element	Description
`Facility`	Syslog facility to use.

```
<Logging>
  <Target>
    <Protocol>file</Protocol>
    <FileName>
      /var/tmp/amps/logs/\%Y\%m\%d\%H\%M\%S-\%n.log
    </FileName>
    <RotationThreshold>2G</RotationThreshold>
    <Level>trace</Level>
    <Levels>critical</Levels>
  </Target>
  <Target>
    <Protocol>syslog</Protocol>
    <Level>critical</Level>
    <Ident>amps_dma</Ident>
    <Options>LOG_CONS,LOG_NDELAY,LOG_PID</Options>
    <Facility>LOG_USER</Facility>
  </Target>
</Logging>
```

Example 8.1. Logging Example

Chapter 9. State-of-the-World (SOW) Features

The `SOW` section of the configuration file holds the configuration for State-of-the-World topics and features that depend on State-of-the-World topics.

Table 9.1. SOW section configuration elements

Element	Description
`TopicDefinition`	Defines a SOW topic. SOW topic definitions are used directly as a last-value cache, and are required for many of the advanced messaging features in AMPS such as out-of-focus notifications and delta messaging. SOW topic definitions are also the underlying topics for views, aggregates, and conflated topics. `TopicDefinition` configuration is described in Section 9.1.
`View`	Defines a View over one or more SOW topics. A view can perform aggregation, can JOIN multiple topics together. A view can be based on a SOW topic of one message type and project results of a different message type. View configuration is described in Section 9.2.
`ConflatedTopic`	Defines a copy of a SOW topic that receives current value updates at a specified interval, conflating any changes to values that occur between the scheduled updates. ConflatedTopic configuration is described in Section 9.3.

The elements within the SOW section are described in detail in the following sections.

9.1. SOW/TopicDefinition

State of the World (SOW) provides a mechanism for AMPS to persist the most recent `publish` for each message. Notice that AMPS does not require topics to be predeclared: defining topics in this way enables the State of the World for the topic. This configuration is not required to publish messages to the topic.

Table 9.2 contains a listing of the parameters for a `TopicDefinition` section in the `SOW` section of an AMPS configuration file.

Table 9.2. SOW/TopicDefinition

Element	Description
`FileName`	The file where the State of the World data will be stored. This element is required for State of the World topics with a `Durability` of `persistent` (the default) because those topics are persisted to the filesystem. This is not required for State of the World topics with a durability of `transient`.
`MessageType`	Type of messages to be stored. To use AMPS generated SOW keys, the message type specified must support content filtering so that AMPS can determine the SOW key for the message. In this release, AMPS loads these message types that support content filtering: `fix`, `nvfix`, `json`, `bson`, and `xml`.

Element	Description
	The `binary` message type does not support content filtering. This message type does not support content filtering, so this message type can only be used for a SOW when publishers use explict keys.
`Topic`	The name of the SOW topic - all unique messages (see `Key`) on this topic will be stored in a topic-specific SOW database.
`Key`	Specifies an XPath within each message that AMPS will use to determine whether a message is unique. This element can be specified multiple times to create a composite key. A SOW topic can have either a key determined by AMPS, or publishers can provide the SOW key for a message with each message. 60East recommends having AMPS determine the key unless your application has specific needs that make this impractical. AMPS automatically creates a hash index for the SOW key.
`HashIndex`	AMPS provides the ability to do fast lookup for SOW records based on specific fields. When one or more `HashIndex` elements are provided, AMPS creates a hash index for the fields specified in the element. These indexes are created on startup, and are kept up to date as records are added, removed, and updated. The `HashIndex` element contains a `Key` element for each field in the hash index. AMPS uses a hash index when a query uses exact matching for all of the fields in the index. AMPS does not use hash indexes for range queries or regular expressions. AMPS automatically creates a hash index for the SOW key.
`RecoveryPoint`	For SOW topics that are covered by the transaction log, the point from which to recover the SOW if the SOW file is removed, or if the SOW topic has `transient` duration. This configuration item allows two values: • `epoch` recovers the SOW from the beginning of the transaction log • `now` recovers the SOW from the current point in the transaction log Defaults to `epoch`.
`Index`	AMPS supports the ability to precreate memo indexes for specific fields using the `Index` configuration option. When one or more `Index` elements are provided, AMPS creates memo indexes for any field specified in an `Index` element on startup, before a query that uses that field runs. Otherwise, AMPS indexes each field the first time a query

Element	Description
	uses the field. Adding one or more `Index` configurations to a `TopicDefinition` can improve retrieval performance the first time a query that contains the indexed fields runs for large SOW topics.
`RecordSize`	Size (in bytes) of a SOW record for this topic. Default: `512`
`InitialSize`	Initial size (in records) of the SOW database file for this topic. Default: `2048`
`IncrementSize`	Number of records to expand the SOW database (for this topic) by when more space is required. Default: `1000`
`Expiration`	Time for how long a record should live in the SOW database for this topic. The expiration time is stored on each message, so changing the expiration time in the configuration file will not affect the expiration of messages currently in the SOW. AMPS accepts interval values for the Expiration, using the interval format described in the AMPS Configuration Guide section on units, or one of the following special values: • A value of `disabled` specifies that AMPS will not process SOW expiration for this topic, regardless of any expiration value set on the message. In this case, AMPS saves the expiration for the message, but does not process it. The value must be set to `disabled` (the default) if `History` is enabled for this topic. • A value of `enabled` specifies that AMPS will process SOW expiration for this topic, with no expiration set by default. Instead, AMPS uses the value set on the individual messages (with no expiration set for messages that do not contain an expiration value). Default: `disabled` (never expire)
`KeyDomain`	The seed value for `SowKeys` used within the topic. The default is the topic name, but it can be changed to a string value to unify `SowKey` values between different topics. For example, if your application has a `ShippingAddress` SOW and a `CreditRating` SOW that both use `/customerID` as the SOW key, you can use a `KeyDomain` to ensure that the generated `SowKey` for a given `/customerId` is identical for both SOW topics. This does not affect how AMPS processes the SOW topics, but can make correlating information from different SOW topics easier in your application. Default: the name of the SOW topic

Element	Description
`Durability`	Defines the data durability of a SOW topic. SOW databases listed as `persistent` are stored to the file system, and retain their data across instance restarts. Those listed as `transient` are not persisted to the file system, and are reset each time the AMPS instance restarts. Default: `persistent` Valid values: `persistent` or `transient` Synonyms: `Duration` is also accepted for this parameter for backward compatibility with configuration prior to 4.0.0.1
`History`	Enable historical query for this SOW. This element contains a `Window` and `Granularity` element. When the `History` element is present, historical query is enabled for this sow. Otherwise, AMPS does not enable historical query and does not store the historical state of the SOW. `Expiration` must be `disabled` when `History` is enabled.
`Window`	For a historical SOW, the length of time to store history. For example, when the value is `1w`, AMPS will store one week of history for this SOW. Used within the `History` element. Default: By default, AMPS does not expire historical SOW data.
`Granularity`	For a historical SOW, the granularity of the history to store. In many cases, it is not necessary for AMPS to store all of the updates to the SOW. This parameter sets the resolution at which you can query history. For example, with a granularity of `1m`, AMPS will store the state of an updated messages no more frequently than once a minute. Used within the `History` element.

An example of a SOW configuration looks like the following:

```
<SOW>

  <!-- Simple SOW topic definition -->
  <TopicDefinition>
    <Topic>orders</Topic>
    <Key>/orderId</Key>
    <MessageType>nvfix</MessageType>
    <FileName>./sow/%n.sow</FileName>
  </TopicDefinition>

  <!-- SOW with hash indexes -->
  <TopicDefinition>
    <Topic>customers</Topic>
    <Key>/customerId</Key>
    <MessageType>json</MessageType>
    <FileName>./sow/%n.sow</FileName>
    <HashIndex>
      <Key>/customerName</Key>
    </HashIndex>
    <HashIndex>
      <Key>/zipCode</Key>
      <Key>/customerType</Key
    </HashIndex>
  </TopicDefinition>

  <!-- Historical SOW -->
  <TopicDefinition>
    <Topic>catalog</Topic>
    <Key>/sku</Key>
    <MessageType>json</MessageType>
    <FileName>./sow/%n.sow</FileName>
    <History>
       <Window>7d</Window>
       <Granularity>15m</Granularity>
    </History>
  </TopicDefinition>
</SOW>
```

Example 9.1. SOW Topic Configuration

9.2. SOW/ViewDefinition

Table 9.3 contains a listing of the parameters for a `ViewDefinition` section in the `SOW` section of an AMPS configuration file.

Table 9.3. SOW/ViewDefinition

Element	Definition
`FileName`	File location to store view data.
`MessageType`	One of the message types configured for the instance. AMPS includes `fix`, `xml`, `nvfix`, `json`, and `bson` message types. You can also use any custom message type defined for the configuration file, provided that the message type supports views. Notice that the `binary` message type does not specify a fixed format for the message contents, so that message type cannot be used in a view.
`Topic`	Defines the topic name for this view.
`UnderlyingTopic`	Defines the SOW topic or topics on which this view is based. This element can contain a single topic name, or any number of `Join` elements.

Element	Definition
`MessageType`	The message type of the view. This does not need to be the same type as any of the topics in the aggregation, but does need to be a message type that supports views.
`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 SQL GROUP BY clause.
`KeyDomain`	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.
`Join`	Within an `UnderlyingTopic`, each `Join` specifies two topics to join together to create the view, as well as the relationship between those topics. An `UnderlyingTopic` can have any number of `Join` specifications. For more information on `Join` specifications, see the *AMPS User Guide*.

```
<SOW>
  <TopicDefinition>
    <Topic>/ett/order</Topic>
    <MessageType>fix</MessageType>
    <Key>/orderId</Key>
  </TopicDefinition>
  <ViewDefinition>
    <FileName>./sow/%n.view.sow</FileName>
    <MessageType>nvfix</MessageType>
    <Topic>TOTAL_VALUE</Topic>
    <UnderlyingTopic>/ett/order</UnderlyingTopic>
    <Projection>
      <Field>/109</Field>
      <Field>SUM(/14 * /6) AS /71406</Field>
    </Projection>
    <Grouping>
      <Field>/109</Field>
    </Grouping>
  </ViewDefinition>
</SOW>
```

Example 9.2. View Example

9.3. SOW/ConflatedTopic

AMPS provides the ability to create ongoing snapshots of a SOW topic, called *conflated topics* (also called *topic replicas* in previous releases of AMPS). Topic replicas are updated on an interval, and store a snapshot of the current state of the world at each interval. This helps to manage bandwidth to clients that do not act on each update, such as a client UI that refreshes every second rather than with every update.

For compatibility with previous versions of AMPS, AMPS allows you to use `TopicReplica` as a synonym for `ConflatedTopic`.

Table 9.4. SOW/ConflatedTopic Parameters

Element	Description
`Topic`	String used to define the name of the conflated topic. While AMPS doesn't enforce naming conventions, it can be convenient to name the conflated topic based on the underlying topic name. For example, if the underlying topic is `orders`, it can be convenient to name the conflated topic `orders-C`.
`UnderlyingTopic`	String used to define the SOW topic which provides updates to the conflated topic. This must exactly match the name of a SOW topic.
`MessageType`	The message format of the underlying topic. This `MessageType` must be the `MessageType` of the provided `UnderlyingTopic`.
`Interval`	The frequency at which AMPS updates the data in the conflated topic. Default: `5 seconds`
`Filter`	Content filter that is applied to the underlying topic. Only messages that match the content filter are stored in the conflated topic.

```
<ConflatedTopic>
 <Topic>FastPublishTopic-C</Topic>
 <FileName>./sow/%n.sow</FileName>
 <MessageType>nvfix</MessageType>
 <UnderlyingTopic>FastPublishTopic</UnderlyingTopic>
 <Interval>5s</Interval>
 <Filter>/region = 'A'</Filter>
</ConflatedTopic>
```

Chapter 10. 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 10.1 contains a listing of the parameters for the `Destination` section in the `Replication` section of an AMPS configuration file.

Table 10.1. Replication Destination

Element	Description
`Destination`	Required parent tag, which defines a unique replication target.
`SyncType`	Defines how synchronization of `ack` messages is handled, either `sync` or `async`.
`Transport`	The message type and URI where messages will be replicated. Requires a `Type`, which must be "amps-replication", and one or more `InetAddr` elements. AMPS supports multiple `Transport` items within a `Destination`. When multiple Transports are provided, AMPS interprets these as transports for redundant servers, listed in priority order. If AMPS cannot connect to any of the internet addresses in a transport, AMPS tries the next `Transport`, in the order in which the `Transport` items appear in the file. When AMPS has tried all of the `Transport` items, AMPS tries again at the beginning of the list of transports. To provide failover, use multiple `InetAddr` elements within a single `Transport` for servers that can use the same `Authenticator` context (that is, the same credentials provided with the same authentication scheme). Use multiple `Transport` elements if the failover servers require different authentication. **`Type`** The `Type` of a replication destination must always be `amps-replication`. **`InetAddr`** A Transport for a replicaiton destination requires one or more InetAddr elements. When a single `InetAddr` element is present, AMPS connect to that address for replication. When more than one `InetAddr` element is present, AMPS uses the list of addresses as a prioritized list of failover servers to provide high availability. The list is in priority order, with the most preferred server at the beginning of the list. Each time AMPS needs to make a connection for this `Destination`, AMPS starts with the first address in the list and tries each address in order until a connection succeeds. If no connection succeeds, AMPS waits for a timeout period and then either moves to the next `Transport` (if more than one `Transport` is present in the destination) or starts again with the first address in the list. Each time AMPS tries all of the addresses in the list without a successful connection,

Element	Description
	AMPS increases the timeout period between tries, up to a maximum timeout. The first time through the list, upon startup, AMPS gives addresses extra time, up to 60 seconds, to connect successfully. **Authenticator** A `Transport` element within a `Destination` may contain an `Authenticator` element, which specifies a module that provides credentials to use when connecting to the destination. All of the `InetAddr` elements specified within a `Transport` use the same `Authenticator`.
`Name`	The name of the destination. This name appears in the AMPS logs when AMPS logs a message about this destination.
`Topic`	Defines the topic name to replicate. Requires a `Name` and `MessageType`. See the following table (Replication Destination : Topic Definition) for details.
`PassThrough`	Specifies source instances to pass through to this destination. The value of this element is a regular expression which is matched against the group name of the instance that sent the replication message to this instance. When the regular expression matches, the replication message is eligible for passthrough, and will be sent to the destination if the `Topic` specifications match the message.
`Compression`	Specifies whether to use compression for this destination. When set to `enabled`, AMPS compresses traffic to this destination. Default: `disabled`

A replication destination can contain any number of Topic definition elements

Table 10.2. Replication Destination : Topic Definition

Element	Description
`Name`	The name of the topic to replicate. The `Name` can be either a literal topic name or a regular expression. When `Name` is a literal topic, a topic with that name and the specified message type must be captured in a transaction log. When `Name` is a regular expression, only topics that match the expression., match the message type, and are present in a transaction log are replicated.
`MessageType`	The message type of the topic to replicate.
`Filter`	A content filter to apply to the topics. When present, only messages that match the filter are replicated. This filter follows the standard AMPS filter syntax.

```
<Replication>
  <Destination>
    <Name>amps-2</Name>
    <Topic>
      <Name>ORDER_STATE-Replication</Name>
      <MessageType>xml</MessageType>
    </Topic>
    <Topic>
      <Name>REFERENCE_INFO-.*</Name>
      <MessageType>json</MessageType>
      <Filter>/state = 'published'</Filter>
    </Topic>
    <SyncType>sync</SyncType>
    <Compression>enabled</Compression>
    <Transport>
      <Type>amps-replication</Type>
      <InetAddr>remote.example.com:19005</InetAddr>
      <InetAddr>remote-backup.example.com:19080</InetAddr>
      <Authenticator>
         <Module>my-credentials-store-module</Module>
      </Authenticator>
    </Transport>
    <Passthrough>NYC</Passthrough>
  </Destination>
</Replication>
```

Example 10.1. Replication Example

Chapter 11. Transaction Log

AMPS includes the ability to record and replay messages. This capability can be used by applications for durable subscriptions, reliable publish, and historical replay. The AMPS transaction log is also the foundation of the high availability features in AMPS. To enable message recording and replay, configure a `TransactionLog` to keep a journal of messages published to an AMPS instance. The *Transactional Messaging and Bookmark Subscriptions* chapter in the AMPS User Guide covers how to use the transaction log for historical replay, durable publish, and durable subscriptions. The *Replication and High Availability* chapter in the *AMPS User Guide* covers the use cases where a `TransactionLog` can be used to maximize the up-time of your AMPS instance.

Table 11.1. TransactionLog Configuration Parameters

Element	Description
`JournalDirectory`	Filesystem location where journal files will be stored.
`JournalArchiveDirectory`	File system location where journal files are archived.
`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: 1GB. Minimum: 10M*
`Topic`	The topic to include in the transaction log. When no `Topic` is specified, AMPS initializes transaction log management for the instance, but does not persist messages.If a `Topic` is specified, then all messages which match exactly the specified topic or regular expression will be included in the transaction log. If you want all topics of a specific message type to be persisted, use the regular expression `.*` for the name of the topic. Multiple `Topic` elements can be included in a `TransactionLog` element.
`FlushInterval`	The interval at which messages will be flushed the journal file during periods of slow activity. *Default: 100ms Maximum: 100ms Minimum: 30us*
`MetadataIndexing`	Specifies whether to create journal index files for the journal. When set to `persistent`, AMPS creates journal index files. When set to `transient`, AMPS does not create journal index files. *Default: persistent*
`O_DIRECT`	Where supported, `O_DIRECT` will perform DMA directly from/to physical memory to a userspace buffer. Having this enabled can improve AMPS performance, however not all devices support `O_DIRECT`. *Default: enabled.*
DEPRECATED `BatchSize`	*This element is no longer necessary in releases of AMPS 4.0 and greater. If this element is present in the configuration, AMPS emits a deprecation warning and ignores the configured value.*

Example 11.1 demonstrates 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 going to contain only those messages which match 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.

```
<AMPSConfig>
...

  <TransactionLog>
    <JournalDirectory>./amps/journal/</JournalDirectory>
    <PreallocatedJournalFiles>1</PreallocatedJournalFiles>
    <MinJournalSize>10MB</MinJournalSize>
    <Topic>
      <Name>orders</Name>
      <MessageType>nvfix</MessageType>
      <Filter>/price > 5</Filter>
    </Topic>
    <FlushInterval>40ms</FlushInterval>
  </TransactionLog>

...
</AMPSConfig>
```

Example 11.1. Transaction Log Configuration Example

Chapter 12. Authentication

The `Authentication` element specifies the module to use for validating user identity. AMPS allows you to set the default `Authentication` for the instance as a whole, and also to set the `Authentication` on each `Transport` individually.

`Authentication` elements are not required. The instance authentication defaults to using the `amps-default-authentication-module` if no `Authentication` element is specified for the instance. An individual `Transport` defaults to using the instance `Authentication` if no `Authentication` element is provided for that `Transport`.

Table 12.1. Authentication Parameters

Name	*Description*
`Module`	The element specifies the name of the module that will be used for authentication. The value of this element must be the name of an authentication module loaded in the Modules section of the configuration file or one of the authentication modules that AMPS loads by default. By default, AMPS loads the authentication modules listed in Table 12.2.
`Options`	A list of supported features for the implemented library. AMPS allows you to pass options to the module by specifying elements within the `Options` element. The exact options that the module requires, if any, are determined by the creator of the module.

AMPS loads the following authentication modules by default:

Table 12.2. AMPS default authentication modules

Module Name	Policy
`amps-default-authentication-module`	Authenticate any user, regardless of the credentials provided. Does not provide the user name to AMPS.
`amps-default-no-authentication-module`	Do not authenticate any user.

Chapter 13. Entitlement

The `Entitlement` element specifies the module to use for validating permissions to resources within AMPS. AMPS allows you to set the default `Entitlement` for the instance as a whole, and also to set the `Entitlement` on each Transport individually.

`Entitlement` elements are not required. The instance authentication defaults to using the `amps-default-entitlement-module` if no `Entitlement` element is specified for the instance. An individual `Transport` defaults to using the instance `Entitlement` if no `Entitlement` element is provided for that `Transport`.

Table 13.1. Entitlement Parameters

Name	*Description*
`Module`	The element specifies the name of the module that will be used for entitlement. The value of this element must be the name of an entitlement module loaded in the Modules section of the configuration file or one of the entitlement modules that AMPS loads by default. By default, AMPS loads the entitlement modules listed in Table 13.2.
`Options`	A list of options to provide to the module for this instance of the module. AMPS allows you to pass options to the module by specifying elements within the `Options` element. The exact options that the module requires, if any, are determined by the creator of the module.

AMPS loads the following entitlement modules by default:

Table 13.2. AMPS default entitlement modules

Module Name	Policy
`amps-default-entitlement-module`	Allow all permissions to every user.
`amps-default-no-entitlement-module`	Deny all permissions to every user.

Chapter 14. Actions

AMPS includes the ability to perform administrative tasks in response to Linux signals or on a set schedule. The `Actions` element allows you to specify these actions and when they occur.

The `Actions` element contains one or more `Action` elements. An `Action` element contains an `On` element, which tells AMPS when to perform the task, and a `Do` element, which tells AMPS what task to perform.

14.1. Running an Action on a Schedule

AMPS provides the `amps-action-on-schedule` module for running actions on a specified schedule.

The options provided to the module define the schedule on which AMPS will run the actions in the Do element.

Table 14.1. Parameters for Scheduling Actions

Parameter	Description
`Every`	Specifies a recurring action that runs whenever the time matches the provided specification. Specifications can take three forms: • *Timer action*. A specification that is simply a duration, such as `4h` or `1d`, creates a timer action. AMPS starts the timer when the instance starts. When the timer expires, AMPS runs the action and resets the timer. • *Daily action*. A specification that is a time of day, such as `00:30` or `17:45`, creates a daily action. AMPS runs the action every day at the specified time. AMPS uses a 24 hour notation for daily actions. • *Weekly action*. A specification that includes a day of the week and a time, such as `Saturday at 11:00` or `Wednesday at 03:30` creates a weekly action. AMPS runs the action each week on the day specified, at the time specified. AMPS uses a 24 hour notation for weekly actions. AMPS accepts both local time and UTC for time specifications. To use UTC, append a `Z` to the time specifier. For example, the time specification `11:30` is 11:30 AM local time. The time specification `11:30Z` is 11:30 AM UTC.
`Name`	The name of the schedule. This name appears in log messages related to this schedule. Default: `unknown`

14.2. Running an Action in Response to a Signal

AMPS provides the `amps-action-on-signal` module for running actions when AMPS receives a specified signal.

The module requires the `Signal` parameter:

Table 14.2. Parameters for Responding to Signals

Parameter	Description
`Signal`	Specifies the signal to respond to. This module supports the standard Linux signals. Configuring an action uses the standard name of the signal. For example, to configure an action to `SIGUSR1`, the value for the `Signal` element is `SIGUSR1`. To configure an action for `SIGHUP`, the value for the `Signal` element is `SIGHUP` and so on. AMPS reserves `SIGQUIT` for producing minidumps, and does not allow this module to override `SIGQUIT`. AMPS registers actions for several signals by default. See the section called “Default Signal Actions” for details.

Actions can be used to override the default signal behavior for AMPS.

Default Signal Actions

By default, AMPS registers the following actions for signals.

Table 14.3. Default Actions

On Event	Action
`SIGUSR1`	`amps-action-do-disable-authentication`
`SIGUSR1`	`amps-action-do-disable-entititlement`
`SIGUSR2`	`amps-action-do-enable-authentication`
`SIGUSR2`	`amps-action-do-enable-entitlement`
`SIGINT`	`amps-action-do-shutdown`
`SIGTERM`	`amps-action-do-shutdown`
`SIGHUP`	`amps-action-do-shutdown`

The actions in the table above can be be overriden by creating an explicit action in the configuration file.

Notice that AMPS also reserves `SIQUIT` to perform the action `amps-action-do-minidump`. This behavior is reserved, and cannot be overriden.

14.3. Running an Action on Startup or Shutdown

AMPS includes modules to run actions when AMPS starts up or shuts down.

The `amps-action-on-startup` module runs actions as the last step in the startup sequence. The `amps-action-on-shutdown` module runs actions as the first step in the AMPS shutdown sequence.

In both cases, actions run in the order that the actions appear in the configuration file.

14.4. Rotate Log Files

AMPS provides the following module for rotating log files. AMPS loads this module by default:

Table 14.4. Managing Logs

Module Name	Does
`amps-action-do-rotate-logs`	Rotates logs that are older than a specified age, for log types that support log rotation. Rotating a log involves closing the log and opening the next log in sequence. AMPS will use the name specifier provided in the AMPS configuration for the new log file. This may overwrite the current log file if the specifier results in the same name as the current log file.

This module requires an `Age` parameter that specifies the age of the log files to process, as determined by the last message written to the file.

Table 14.5. Parameters for Rotating Log Files

Parameter	Description
`Age`	Specifies the age of files to process. The module processes any file older than the specified `Age`. For example, when the `Age` is `5d`, only files that have been unused for longer than 5 days will be processed by the module. AMPS does not process the current log file, even if it has been inactive for longer than the `Age` parameter. There is no default for this parameter.

14.5. Manage Statistics Files

AMPS provides the following modules for managing statistics. As a maintenance strategy, 60East recommends truncating statistics on a regular basis. This frees space in the statistics file, which will be reused as new statistics are generated. It is generally not necessary to vacuum statistics unless you have changed your retention policy so that less data is retained between truncation operations. With regular truncation, the statistics file will usually stabilize at the correct size to hold the amount of data your application generates between truncation operations.

AMPS loads these modules by default.

Table 14.6. Managing Logs

Module Name	Does
`amps-action-do-truncate-statistics`	Removes statistics that are older than a specified age. This frees space in the statistics file, but does not reduce the size of the file.
`amps-action-do-vacuum-statistics`	Remove unused space in the statistics file to reduce the size of the file. In general, it is not necessary to remove unused space in the statistics file. This operation can be expensive, and query access to the statistics database can be unavailable for an extended period of time if the file is large. If storage space is in high demand, and the interval at which the file is vacuumed has been reduced, removing space from the file can sometimes reduce the space needs. 60East recommends using this action only in long-running AMPS environments where space is at a premium, and scheduling the action during times when it is acceptable for monitoring of the system to be unavailable while the file is processed.

The `amps-action-do-truncate-statistics` module requires an `Age` parameter that specifies the age of the statistics to process.

Table 14.7. Parameters for Managing Statistics

Parameter	Description
`Age`	Specifies the age of the statistics to remove. The module processes any file older than the specified `Age`. For example, when the `Age` is `5d`, the module removes statistics that are older than 5d. There is no default for this parameter.

14.6. Manage Journal Files

AMPS provides the following modules for managing journal files. AMPS loads these modules by default:

Table 14.8. Managing Journals

Module Name	Does
`amps-action-do-archive-journal`	Archives journal files that are older than a specified age to the `JournalArchiveDirectory` specified for the transaction log.
`amps-action-do-compress-journal`	Compresses journal files that are older than a specified age.
`amps-action-do-remove-journal`	Deletes journal files that are older than a specified age.

Each of these modules requires an `Age` parameter that specifies the age of the journal files to process.

Table 14.9. Parameters for Managing Journals

Parameter	Description
`Age`	Specifies the age of files to process. The module processes any file older than the specified `Age`. For example, when the `Age` is `5d`, only files that have been unused for longer than 5 days will be processed by the module. AMPS does not process the current log file, or files that are being used for replay or replication, even if the file has been inactive for longer than the `Age` parameter. There is no default for this parameter.

14.7. Removing Files

AMPS provides the following module for removing files. Use this action to remove error log files that are no longer needed. AMPS loads this module by default. This action cannot be used to safely remove journal files (also known as transaction log files). For those files, use the journal management actions described in Section 14.6.

⚠ This action removes files that match an arbitrary pattern. If the pattern is not specified carefully, this action can remove files that contain important data, are required for AMPS, or are required by the operating system.

⚠ This action cannot be used to safely remove journal files. Use the actions in Section 14.6 to manage journal files.

Table 14.10. Removing Files

Module Name	Does
`amps-action-do-remove-files`	Removes files that match the specified pattern that are older than the specified age. This action accepts an arbitrary pattern, and removes files that match that pattern. While AMPS attempts to protect against deleting journal files, using a pattern that removes files that are crit-

Module Name	Does
	ical for AMPS, for the application, or for the operating system may result in loss of data. The module does not recurse into directories. It skips open files. The module does not remove AMPS journals (that is, files that end with a `.journal` extension), and reports an error if a file with that extension matches the specified `Pattern`. The commands to remove files are executed with the current permissions of the AMPS process.

This module requires an `Age` parameter that specifies the age of the files to remove, as determined by the update to the file. This module also requires a `Pattern` parameter that specifies a pattern for locating files to remove.

Table 14.11. Parameters for Removing Files

Parameter	Description
`Age`	Specifies the age of files to process. The module removes any file older than the specified `Age` that matches the specified `Pattern`. For example, when the `Age` is `5d`, only files that have not modified within 5 days and that match the pattern will be processed by the module. There is no default for this parameter.
`Pattern`	Specifies the pattern for files to remove. The module removes any files that match the specified `Pattern` that have not been modified more recently than the specified `Age`. This parameter is interpreted as a Unix shell globbing pattern. It is *not* interpreted as a regular expression. As with other parameters that use the file system, when the pattern specified is a relative path the parameter is interpreted relative to the current working directory of the AMPS process. When the pattern specified is an absolute path, AMPS uses the absolute path. There is no default for this parameter.

14.8. Manage SOW Contents

The `amps-do-delete-sow` module deletes messages from SOW topics. The module accepts the following options:

Table 14.12. Parameters for Deleting SOW Messages

Parameter	Description
`MessageType`	The MessageType for the SOW topic.

Parameter	Description
	There is no default for this parameter.
`Topic`	The name of the SOW topic from which to delete messages. There is no default for this parameter
`Filter`	Set the filter to apply. If a `Filter` is present, only messages matching that filter will be deleted.

14.9. Create Mini-Dump

AMPS minidumps provide a way for the 60East Technologies engineering team to inspect the state of a running AMPS system.

The `amps-do-minidump` module creates a minidump. This module is typically used with the `amps-action-on-signal` module to provide a way for a developer or administrator to easily create a minidump for diagnostic purposes.

14.10. Manage Security

AMPS provides modules for managing the security features of an instance.

Authentication and entitlement can be enabled or disabled, which is useful for debugging or auditing purposes. You can also reset security and authentication, which clears the AMPS internal caches and gives security and authentication modules the opportunity to reinitialize themselves, for example, by re-parsing an entitlements file.

AMPS loads the following modules by default:

Table 14.13. Security Modules

Module Name	Does
`amps-action-do-disable-authentication`	Disables authentication for the instance.
`amps-action-do-disable-entitlement`	Disables entitlement for the instance.
`amps-action-do-enable-authentication`	Enables authentication for the instance.
`amps-action-do-enable-entitlement`	Enables entitlement for the instance.
`amps-action-do-reset-authentication`	Resets authentication by clearing AMPS caches and reinitializing authentication
`amps-action-do-reset-entitlement`	Resets entitlement by clearing AMPS caches and reinitializing entitlement

These modules require no parameters.

14.11. Manage Transports

AMPS provides modules that can enable and disable specific transports.

Table 14.14. Transport Action Modules

Module Name	Does
`amps-action-do-enable-transport`	Enables a specific transport.
`amps-action-do-disable-transport`	Disables a specific transport.

These modules accept the following options:

Table 14.15. Parameters for Managing Transports

Parameter	Description
`Transport`	The `Name` of the transport to enable or disable. If no `Name` is provided, the module affects all transports.

14.12. Manage Replication

AMPS provides modules for downgrading replication destinations that fall behind and upgrading them again when they catch up.

Table 14.16. Replication Modules

Module Name	Does
`amps-action-do-downgrade-replication`	Downgrades replication connections from synchronous to asynchronous if the age of the last acknowledged message is older than a specified time period.
`amps-action-do-upgrade-replication`	Upgrades previously-downgraded replication connections from asynchronous to synchronous if the age of the last acknowledged message is more recent than a specified time period.

The modules determine when to downgrade and upgrade based on the age of the oldest message that a destination has not yet acknowledged. When using these modules, it is important that the thresholds for the modules are not set too close together. Otherwise, AMPS may repeatedly upgrade and downgrade the connection when the destination is consistently acknowledging messages at a rate close to the threshold values. To avoid this, 60East recommends that the `Age` set for the upgrade module is 1/2 of the age used for the downgrade module.

The `amps-action-do-downgrade-replication` module accepts the following options:

Table 14.17. Parameters for Downgrading Replication

Parameter	Description
`Age`	Specifies the maximum message age at which AMPS downgrades a replication destination to `async`. When this action runs, AMPS downgrades any destination for which the oldest unacknowledge message is older than the specified `Age`. For example, when the `Age` is `5m`, AMPS will downgrade any destination where a message older than 5 minutes has not been acknowledged. There is no default for this parameter.
`GracePeriod`	The approximate time to wait after start up before beginning to check whether to downgrade links. The `GracePeriod` allows time for other AMPS instances to start up, and for connections to be established between AMPS instances.

The `amps-action-do-upgrade-replication` module only applies to destinations configured as `sync` that have been previously downgraded. The module accepts the following options:

Table 14.18. Parameters for Upgrading Replication

Parameter	Description
`Age`	Specifies the maximum message age at which a previously-downgraded destination will be upgraded to `sync` mode. When this action runs, AMPS upgrades any destination that has been previously downgraded where the oldest unacknowledged message to AMPS is more recent than time value specified in the `Age` parameter. For example, if a destination has been downgraded to `async` mode and the `Age` is `2m`, AMPS will upgrade the destination when the oldest unacknowledged message to that destination is less than 2 minutes old. There is no default for this parameter.
`GracePeriod`	The approximate time to wait after start up before beginning to check whether to upgrade links. The `GracePeriod` allows time for other AMPS instances to start up, and for connections to be established between AMPS instances.

14.13. Shut Down AMPS

The `amps-action-do-shutdown` module shuts down AMPS. This module is registered as the default action for several Linux signals, as described in the section called "Default Signal Actions".

Table 14.19. Do Nothing Module

Module Name	Does
`amps-action-do-shutdown`	Shuts down AMPS.

14.14. Do Nothing

The amps-action-do-nothing module does not modify the state of AMPS in any way. The module simply logs that it was called.

The module provides a convenient way of testing schedule specifications or signal handling without requiring further configuration.

Table 14.20. Do Nothing Module

Module Name	Does
`amps-action-do-nothing`	Takes no action.

14.15. Action Configuration Examples

Archive Files Older Than One Week, Every Saturday

The listing below asks AMPS to archive files older than 1 week, every Saturday at 12:30 AM:

```
<Actions>
      <Action>
        <On>
          <Module>amps-action-on-schedule</Module>
          <Options>
            <Every>Saturday at 00:30</Every>
            <Name>Saturday Night Fever</Name>
          </Options>
        </On>
        <Do>
          <Module>amps-action-do-archive-journal</Module>
          <Options>
              <Age>7d</Age>
          </Options>
        </Do>
      </Action>
```

```
   </Actions>
```

Disable and Re-enable Security on Signal

The listing below disables authentication and entitlement when AMPS receives on the `USR1` signal. When AMPS receives the `USR2` signal, AMPS re-enables authentication and entitlement. This configuration is, in effect, the configuration that AMPS installs by default for these signals:

```
<Actions>
      <Action>
        <On>
          <Module>amps-action-on-signal</Module>
          <Options>
            <Signal>SIGUSR1</Signal>
          </Options>
        </On>
        <Do>
          <Module>amps-action-do-disable-authentication</Module>
        </Do>
        <Do>
          <Module>amps-action-do-disable-entitlement</Module>
        </Do>
      </Action>
      <Action>
        <On>
          <Module>amps-action-on-signal</Module>
          <Options>
            <Signal>SIGUSR2</Signal>
          </Options>
        </On>
        <Do>
          <Module>amps-action-do-enable-authentication</Module>
        </Do>
        <Do>
          <Module>amps-action-do-enable-entitlement</Module>
        </Do>
      </Action>
   </Actions>
```

Index