Simple Asynchronous Messaging

Introduction

This extension provides access to the functionality of messaging and queueing systems, such as the IBM WebSphere MQSeries family of products, from PHP scripts. The interface is designed to make it extremely simple to do the more commonly required tasks such as deliver simple text messages to queues while still allowing skilled users to do more complex messaging operations. For many users the complexities of setting up numerous options can be simply ignored.

The SAM extension is a framework that provides a very simple API that can be used to access a number of messaging middleware systems. Currently the package includes built-in support for the MQTT (MQ Telemetry Transport) messaging protocol and support for the IBM Messaging and Queuing middleware products. SAM is designed to be readily extended to support other messaging systems and extension modules may be written in C or PHP.

Installing/Configuring

Requirements

The SAM extension interfaces to the IBM Messaging and Queuing middleware products using a set of libraries and some client side code referred to as XMS. This package is available as a free download in the guise of IBM support pack IA94. There is a description of this package and download links in the article » Introducing XMS - The IBM Message Service API.

If you intend to use SAM to access the Messaging and Queuing infrastructure within WebSphere MQ then you will also need to have installed a local MQ queue manager or installed the WebSphere MQ clients package. The clients package is freely available as a support pack (» MQC6).

If you are only aiming to experiment with sending messages to and from WebSphere Application Server queues using the WebSphere Platform Messaging protocol (WPM) then you do not need to install the MQC6 package.

After installing these packages you will need to ensure the XMS binary and, if you are using it, the MQ client bin directory are included in the PATH environment variable so that Apache and PHP can find the dependent .DLLs/libraries.

Installation

The SAM framework and MQTT support can be built and used without any other prerequisites. Support for protocols other than MQTT is provided via a set of libraries and some client side code referred to as XMS.

If you only intend to use the built-in MQTT support then you can build and configure SAM as an extension or simply refer to "php_sam.php" with a "requires" or "requires_once" clause in your PHP script. In this case you need only install the code without building the extension using the pear installer:

```
pecl install -B SAM
```

Linux installation steps

The sam extension is supplied as a PECL module, which you should be able to download and install in one step as follows:

```
pecl install sam
```

(Depending on your php environment, you will probably need to be root to do this.)

Make sure that the module is loaded by PHP, by adding following line to *php.ini*:

```
extension=sam.so
```

If you intend to use the XMS support to access the IBM Messaging and Queuing family you must also enable the SAM XMS extension.

```
extension=sam_xms.so
```

If you cannot use the PEAR installer, you can download the extension and build it manually:

```
pear download sam          #downloads sam-<version>.tgz
tar -xzf sam-<version>.tgz
cd sam-<version>
phpize
./configure
make
make install               #you may need to be root for this step
```

To work with the very latest source, you'll need to extract it from cvs and build manually as above.

Windows installation steps

You will probably need to build the sam extension for Windows as there are only a limited range of pre-built binaries available from the SAM website. The extension can be built using the standard Windows extension build procedures.

You will need the PHP source tree for the version of PHP you wish to build the SAM extension against which you can obtain from php.net. This should be unpacked into a working directory of your choice.

You will also need the libraries and headers used by PHP extensions available from http://www.php.net/extra/win32build.zip and this should be unzipped so that is in your working directory.

You should have something like:

```
c:\php-build\-
             |
             |---php-5.0.5--|---build
             |              |---ext
             |              |--- ...
             |
             |---win32build--|---bin
                             |---include
                             |---lib
```

You will need a compiler such as the free version of Visual Studio C++ Express from the Microsoft web site. Also you need the Microsoft Windows Platform SDK which again can be downloaded from the Microsoft web site.

Obtain the SAM extension source using pear (pecl download sam) or by using CVS and copy the files to a new "sam" directory under the "ext" directory in your PHP source tree.

To build the extension open a build environment window by going to the start menu->all programs->microsoft platform SDK for windows-> open build environment window->windows 200 build environment-> set windows 2000 build environment (retail)

This should open a command prompt with all the environment variables set up to access the platform SDK etc. You then need to set the environment variables for Visual Studio by issuing the command "vcvars32.bat" in the window.

Change directory to your working directory e.g. cd c:\php-build. Then make sure the win32build tools are accessible by adding them to the PATH environment variable:

```
set PATH=..\win32build\bin;%PATH%
```

Run the buildconf.bat command. This should rebuild the configure.js file.

Run the cscript command with the appropriate options. To build just the SAM extension framework and MQTT support use:

```
cscript /nologo configure.js --with-sam
```

To build the SAM framework and the XMS support use:

```
cscript /nologo configure.js --with-sam --with-sam_xms="c:\program
files\ibm\xms"
```

The additional parameter passed for sam_xms is the installation path to the XMS libraries and runtime that were installed as described under prerequisites at the top of this page.

You can specify whatever other cscript parameters you require to include or exclude items from the php build or select options.

Assuming all has gone well so far you can now finally run a make for the SAM framework!

```
nmake php_sam.dll
```

Also if you are using the XMS support you must make the sam_xms extensions:

```
nmake php_sam_xms.dll
```

If you have used Visual Studio 2005 to build the DLLs please see below for additional steps that must be carried out before proceeding further.

The DLLs created (php_sam.dll and optionally php_sam_xms.dll) can now be copied to the subdirectory appropriate for your PHP set-up. Make sure that the module(s) are loaded by PHP, by adding following line to *php.ini*:

```
extension=php_sam.dll
```

If you intend to use the XMS support to access the IBM Messgaing and Queuing family you must also enable the SAM XMS extension.

```
extension=php_sam_xms.dll
```

Additional steps for Visual Studio 2005

If you build the SAM extension with the Microsoft Visual Studio 2005 compiler and tools you need to perform an additional step in the build process to ensure the *php_sam.dll* is able to link with the C runtime libraries at runtime. This step includes the dependancy manifest into the DLL. Switch to the directory where the *php_sam.dll* has been generated (usually Release_TS or Debug_TS below the php source directory) and issue the following magic incantation:

```
mt.exe -manifest php_sam.dll.manifest -outputresource:php_sam.dll;2
```

If you are using the XMS capabilities you will need to do the same with the SAM XMS DLL:

```
mt.exe -manifest php_sam_xms.dll.manifest -outputresource:php_sam_xms.dll;2
```

If you build the SAM extension using the compiler and libaries from Microsoft Visual Studio 2005 you will also need to ensure that the runtime components are installed on the system on which you intend to use SAM. This can be accomplished by installing Visual Studio 2005 or by using the freely distributable » runtime package.

Runtime Configuration

Protocol support and mapping

The SAM framework can be extended to support other messaging protocols and connection mechanisms. To add support for a new protocol or connection library a support class has to be defined, either as a C extension or as a PHP script, and a "factory" script must be created. The support class must implement all the methods of the SAMConnection class though it should not inherit from SAMConnection. The factory script will be called by the SAM framework to create an instance of the implemented class. The way SAM chooses which factory to call is based on the protocol specified as the first parameter of the "connect" call.

By default the built-in MQTT support will be used if a connect call specifies a protocol of SAM_MQTT ("mqtt"), for any other protocol SAM will attempt to use the XMS support extension. To add support for additional protocols or to modify the default behavior entries may be added to *php.ini* in the [sam] section. The default mapping is equivalent to the following entries:

```
[sam]
sam.factory.mqtt=mqtt
sam.factory.wmq=xms
sam.factory.wmq:client=xms
sam.factory.wmq:bindings=xms
sam.factory.wpm=xms
sam.factory.rtt=xms
```

As can be seen from these examples the entries take the form of "sam.factory.pppp=xxx" where pppp is the protocol string specified on the connect call and xxx is a factory suffix. Note: SAM defines constants for these protocol strings such that SAM_WMQ=wmq, SAM_WPM=wpm, SAM_RTT=rtt, SAM_MQTT=mqtt, etc.

When identifying the support code to use on a connect call SAM looks up the protocol name in the *php.ini* entries and then invokes a factory script named sam_factory_xxx.php. If no entry is found the support will default to XMS.

Resource Types

This extension has no resource types defined.

Predefined Constants

The constants below are defined by this extension, and will only be available when the extension has either been compiled into PHP or dynamically loaded at runtime.

SAM_AUTO (string)
Automatic behaviour

SAM_BOOLEAN (string)
Type specifier used when setting properties on SAM_Message objects.

SAM_BUS (string)
Connect attribute used to set the name of the enterprise service bus to connect to.

SAM_BYTE (string)
Type specifier used when setting properties on SAM_Message objects.

SAM_BYTES (string)
Message body type descriptor.

SAM_CORRELID (string)
Attribute used on receive, send and remove requests to identify specific messages.

SAM_DELIVERYMODE (string)
Message header property.

SAM_DOUBLE (string)
Type specifier used when setting properties on SAM_Message objects.

SAM_ENDPOINTS (string)
Connect attribute used to define the possible endpoints to connect to.

SAM_FLOAT (string)
Type specifier used when setting properties on SAM_Message objects.

SAM_HOST (string)
Connect attribute used to set the hostname of the required messaging server.

SAM_INT (string)
Type specifier used when setting properties on SAM_Message objects.

SAM_LONG (string)
Type specifier used when setting properties on SAM_Message objects.

SAM_MANUAL (string)
Manual (script controlled) behaviour

SAM_MESSAGEID (string)
Attribute used on receive and remove requests to identify specific messages.

SAM_MQTT (string)
Connect protocol definition for selecting the MQTT (MQ Telemetry Transport) protocol.

SAM_MQTT_CLEANSTART (bool)
Optional connect option to indicate to an MQTT server that all previous connection data for this client should be removed and that subscriptions should be deleted when the client disconnects explicitly or unexpectedly.

SAM_NON_PERSISTENT (string)
Connect attribute value used to request messages are not made persistent on the messaging server.

SAM_PASSWORD (string)
Connect attribute used to define the password to be used for the user account being used to connect to a messaging server that requires authorisation for connections.

SAM_PERSISTENT (string)
Connect attribute value used to request messages are made persistent on the messaging server to protect against loss of messages in the event of failure.

SAM_PORT (string)
Connect attribute used to set the port number on which to communicate with the messaging server.

SAM_PRIORITY (string)
Option name used on send requests to specify a delivery priority value.

SAM_REPLY_TO (string)
Message propery used to specify the queue identity on to which the script expects response or reply messages to be posted.

SAM_RTT (string)
Connect protocol definition for selecting the IBM Realtime Transport protocol for communication with a business integration messaging server.

SAM_STRING (string)
Type specifier used when setting properties on SAM_Message objects.

SAM_TARGETCHAIN (string)
Connection attribute used to set the required target chain identifier.

SAM_TEXT (string)
Message body type descriptor.

SAM_TIMETOLIVE (string)
Message send option name used to specify the length of time a message should be retained in milliseconds.

SAM_TRANSACTIONS (string)
Connection attribute used to set required transactional behaviour. May be set to SAM_AUTO (default) or SAM_MANUAL.

SAM_USERID (string)
Connect attribute used to define the account to being used to connect to a messaging server that requires authorisation for connections.

SAM_WAIT (string)
Receive property used to specify the wait timeout to be used when receiving a message from a queue or subscription.

SAM_WMQ (string)
Connect protocol definition for selecting the IBM WebSphere MQSeries protocol for communication with the desired messaging server.

SAM_WMQ_BINDINGS (string)
Connect protocol definition for selecting the IBM WebSphere MQSeries protocol for communication with a local messaging server.

SAM_WMQ_CLIENT (string)
Connect protocol definition for selecting the IBM WebSphere MQSeries protocol for communication with a remote messaging server.

SAM_WMQ_TARGET_CLIENT (string)
Option name used on send requests to specify the target client mode. This can either be default to 'jms' or 'mq'. The default is 'jms' which means an RFH2 header is sent with the message whereas the 'mq' setting means no RFH2 is included.

SAM_WPM (string)
Connect protocol definition for selecting the IBM WebSphere Platform Messaging protocol for communication with a WebSphere Application Server messaging server.

Examples

Connections

In order to perform any messaging and queueing functions a connection must be established with a messaging server by creating a SAMConnection object and calling its "connect" method, with a set of connection properties, to connect the PHP script to the messaging server. Until such time as the SAMConnection object is destroyed the connection will be maintained and available for use. All SAMConnection objects are destroyed when the PHP script exits.

A set of default properties may be used in connecting to a messaging server but as a minimum the PHP script must specify a protocol to be used.

Example #1 - Creating a connection and connecting to a remote WebSphere MQSeries Messaging Server

```
<?php
$conn = new SAMConnection();
$conn->connect(SAM_WMQ, array(SAM_HOST => 'myhost.mycompany.com',
                              SAM_PORT => 1506,
                              SAM_BROKER => 'mybroker'));
?>
```

Example #2 - Creating a connection and connecting to a remote WebSphere Application Server

```
<?php
$conn = new SAMConnection();
$conn->connect(SAM_WPM, array(SAM_ENDPOINTS =>
'localhost:7278:BootstrapBasicMessaging',
                              SAM_BUS => 'Bus1',
                              SAM_TARGETCHAIN => 'InboundBasicMessaging'));
?>
```

Example #3 - Creating a connection and connecting to an MQTT server

```
<?php
$conn = new SAMConnection();
$conn->connect(SAM_MQTT, array(SAM_HOST => 'myhost.mycompany.com',
                               SAM_PORT => 1883));
```

```
?>
```

Messages

Messages sent to and received from queues are represented by the SAMMessage object. The SAMMessage object encapsulates the body of the message (if one exists) and the header properties associated with the message. A SAMMessage object is either supplied as a parameter to a messaging operation or returned as a result.

Example #4 - Creating a message with a simple text body

```
<?php
$msg = new SAMMessage('This is a simple text message');
?>
```

Messages may have header properties associated with them that provide control over the transport of the message or further information to the receiving application. By default message properties are delivered to the underlying messaging system as strings and in this case they may be set with the following simple syntax:

Example #5 - Setting a text format property using the default syntax

```
<?php
$msg->header->myPropertyName = 'textData';
?>
```

If it is desired to pass type information an alternative syntax may be used where the value and the type hint are passed in an associative array:

Example #6 - Setting a property using a type hint

```
<?php
$msg->header->myPropertyName = array(3.14159, SAM_FLOAT);
?>
```

Properties may also be extracted from the header of a message.

Example #7 - Retrieving a property from a message header

```
<?php
$myProperty = $msg->header->myPropertyName;
?>
```

Messaging operations

All messaging operations are performed through calls to methods on the connection object. To add a message to a queue the "send" method is used, to obtain a message from a queue the "receive" method is used. Other methods provide publish and subscribe functionality and control of transaction boundaries.

Example #8 - Adding a message to a queue and receiving a response

```
<?php
$msg = new SAMMessage('This is a simple text message');
$msg->header->SAM_REPLY_TO = 'queue://receive/test';
$correlid = $conn->send('queue://send/test', $msg);

if (!$correlid) {
 // The Send failed!
 echo "Send failed ($conn->errno) $conn->error";
} else {
 $resp = $conn->receive('queue://receive/test', array(SAM_CORRELID =>
$correlid));
}
?>
```

Publish/Subscribe and suscriptions to topics

SAM allows messages to be sent either to queues or, for WebSphere MQ and WPM, to publish/subscribe topics. A topic desintation is specified to SAM in the usual way, i.e. in the form 'topic://fred', rather than the form 'queue://AQUEUE' used for point to point operation. To use publish/subscribe it is simply necessary to specify the correct broker name on the SAMConnect "connect" call and the desired topic in the destination argument to the SAMConnect "send" and "receive" calls. The PHP interface is otherwise identical to the point to point model.

By default, SAM creates non-durable subscriptions when using publish/subscribe. This means that if a client application is inactive when messages are published to a topic, then it will not receive them when it subsequently restarted. SAM does also allow durable subscriptions to be made to topics when using WPM or WebSphere MQ publish/subscribe. The purpose of these subscriptions is to allow data to be received by a client application even if that client was not active at the time the data was published.

Durable subscriptions are specified by using the SAMConnect "subscribe" call. This method takes the destination topic as an input parameter and returns a subscription identifier that may be used on subsequent "receive" calls. When the subscription is no longer required the SAMConnection "unsubscribe" method should be used to delete the subscription.

Example #9 - Creating a durable subscription to a topic

```
<?php

$subName = $conn->subscribe('topic://A');

if (!$subName) {
  echo "Subscribe failed";
} else {
  # Subscribe was OK
  // ...
}
?>
```

Example #10 - Subscribing to a topic using a WebSphere Platform Messaging (WPM) server

```
<?php
$conn = new SAMConnection();
// Note: For pub/sub on WPM, when connecting the name of a messaging engine
//   to hold the durable subscription (SAM_WPM_DUR_SUB_HOME) must be
specified.
$conn->connect(SAM_WMQ, array(SAM_ENDPOINTS =>
'localhost:7278:BootstrapBasicMessaging',
                              SAM_BUS => 'Bus1',
                              SAM_TARGETCHAIN => 'InboundBasicMessaging'
                              SAM_WPM_DUR_SUB_HOME =>
'MyMachineNode01.server1-Bus1'));

$subName = $conn->subscribe('topic://A');

if (!$subName) {
  echo "Subscribe failed";
} else {
  # Subscribe was OK
  // ...
}
?>
```

Example #11 - Receiving published data using a durable subscription

```
<?php

$msg = $conn->receive($subName);
if ($msg) {
  echo "Received a message OK";
} else {
  echo "The receive failed";
}

?>
```

Example #12 - Deleting a durable subscription to a topic

```
<?php

if (!$conn->unsubscribe($subName)) {
  echo "Unsubscribe failed";
}

?>
```

Error handling

All SAMConnection methods that provide access to messaging operations return **FALSE** if an error occurred in processing the request. In addition the SAMConnection object has two properties, "errno" and "error", that provide respectively the error number and text description of the last error to occur on the connection.

Example #13 - Handling an error from a method that returns no result

```
<?php
if (!$conn->commit()) {
   // The commit failed!
   echo "Commit failed ($conn->errno) $conn->error";
}
?>
```

Example #14 - Handling an error from a method that returns a result

```
<?php
$correlid = $conn->send('queue://send/test', $msg);

if (!$correlid) {
 // The Send failed!
 echo "Send failed ($conn->errno) $conn->error";
} else {
 ...
}
?>
```

SAM Functions

Predefined Classes

SAMConnection

Object representing a connection to a Messaging Server

Constructor

- new SAMConnection - construct a new connection object to allow connection to a messaging infrastructure.

Methods

- commit - a method that commits (successfully completes) an in-flight unit of work.
- connect - a method that connects a PHP script to a messaging server.
- disconnect - a method that disconnects a PHP script from a messaging server.
- isConnected - a method that checks whether a PHP script is connected to a messaging server.
- peek - a method that receives a message from a queue without removing it from the queue.
- peekAll - a method that receives one or messages from a queue without removing them from the queue.
- receive - a method that receives a message from a queue or subscription.
- remove - a method that removes a message from a queue.
- rollback - a method that cancels (rolls back) an in-flight unit of work.
- send - a method that sends a message to a queue or posts to a topic
- setDebug - a method that switches additional debugging output on or off
- subscribe - a method that creates a subscription to one or more topics
- unsubscribe - a method that destroys a subscription to one or more topics

Properties

- errno - the numeric error code for the last encountered error on this connection. This property is set to 0 if the last operation was successful.
- error - the text description for the last encountered error on this connection

SAMMessage

Object representing a message to be sent or received

Constructor

- new SAMMessage - construct a new message.

Properties

- body - the body of the message.
- header - the header properties of the message.

SAMConnection->commit()

SAMConnection->commit() -- Commits (completes) the current unit of work.

Description

SAMConnection

bool **commit** (void)

Calling the "commit" method on a Connection object commits (completes) all in-flight transactions that are part of the current unit of work.

Return Values

This method returns **FALSE** if an error occurs.

Examples

Example #15 - Committing the current unit of work

```
<?php
 if (!$conn->commit()) {
   // The commit failed!
   echo "Commit failed ($conn->errno) $conn->error";
 }
?>
```

See Also

- SAMConnection->rollback()

SAMConnection->connect()

SAMConnection->connect() -- Establishes a connection to a Messaging Server

Description

SAMConnection

bool **connect** (string `$protocol` [, array `$properties`])

Calling the "connect" method on a SAMConnection object connects the PHP script to a messaging server. No messages can be sent or received until a connection is made.

Parameters

Return Values

This method returns **FALSE** if an error occurs.

Examples

Example #16 - Creating a connection to a Messaging Server using the IBM MQSeries protocol (WMQ)

```
<?php

$conn->connect(SAM_WMQ, array(SAM_HOST => 'Myhost.myco.com', SAM_PORT =>
1506, SAM_BROKER => 'MyBroker'));

?>
```

Example #17 - Creating a connection with application transaction control and default host and port values

```
<?php

$conn->connect(SAM_WMQ, array(SAM_BROKER => 'MyBroker', SAM_TRANSACTIONS =>
SAM_MANUAL));
```

```
?>
```

Example #18 - Creating a connection to a Messaging Server using the IBM WebSphere Platform Messaging protocol (WPM)

```
<?php

$conn->connect(SAM_WPM, array(SAM_ENDPOINTS =>
'localhost:7278:BootstrapBasicMessaging',
                              SAM_BUS => 'Bus1', SAM_TARGETCHAIN =>
'InboundBasicMessaging'));

?>
```

See Also

- SAMConnection->isConnected()
- SAMConnection->disconnect()

SAMConnection->__construct()

SAMConnection->__construct() -- Creates a new connection to a Messaging Server

Description

SAMConnection

__construct ( )

Creates a new SAMConnection object.

Examples

Example #19 - Creating a connection object and connecting to a Messaging Server

```
<?php

$conn = new SAMConnection();
$conn->connect(SAM_WMQ, array(SAM_HOST => localhost, SAM_PORT => 1414,
SAM_BROKER => 'bull'));

?>
```

SAMConnection->disconnect()

SAMConnection->disconnect() -- Disconnects from a Messaging Server

Description

SAMConnection

bool **disconnect** (void)

Calling the "disconnect" method on a SAMConnection object disconnects the PHP script from a messaging server. No messages can be sent or received after a connection has been disconnected.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #20 - Disconnecting from a Messaging Server

```
<?php

$conn->disconnect();

?>
```

See Also

- SAMConnection->isConnected()
- SAMConnection->connect()

SAMConnection->errno

SAMConnection->errno -- Contains the unique numeric error code of the last executed SAM operation.

Description

SAMConnection

int *errno*;

Contains the numeric error code of the last executed SAM operation on this connection. If the last operation completed successfully this property contains 0.

Return Values

An integer greater than zero indicates the last error type encountered on the connection. Zero indicates that the last operation on this connection completed successfully.

Examples

Example #21 - Using the error number and description properties

```
<?php
$conn = new SAMConnection();
$conn->connect(SAM_WMQ, array(SAM_HOST => 'localhost', SAM_PORT => 1506));
$msg = new SAMMessage('This is a simple text item');
if (!$conn->send('topic://test', $msg)) {
   // The Send failed!
   echo "Send failed ($conn->errno) $conn->error";
}
?>
```

See Also

- SAMConnection->error

SAMConnection->error

SAMConnection->error -- Contains the text description of the last failed SAM operation.

Description

SAMConnection

string *error*;

Contains the text description of the last failed SAM operation on this connection. If the last operation completed successfully this property contains an empty string.

Return Values

A string containing the text description of the last error type encountered on the connection. An empty string indicates that the last operation on this connection completed successfully.

Examples

Example #22 - Using the error number and description properties

```
<?php
$conn = new SAMConnection();
$conn->connect(SAM_WMQ, array(SAM_HOST => 'localhost', SAM_PORT => 1506));
$msg = new SAMMessage('This is a simple text item');
if (!$conn->send('topic://test', $msg)) {
   // The Send failed!
   echo "Send failed ($conn->errno) $conn->error";
}
?>
```

See Also

- SAMConnection->errno

SAMConnection->isConnected()

SAMConnection->isConnected() -- Queries whether a connection is established to a Messaging Server

Description

SAMConnection

bool **isConnected** (void)

Calling the "isConnected" method on a Connection object will check whether the PHP script is connected to a messaging server. No messages can be sent or received unless a connection has been established with a Messaging server.

Return Values

This method returns **TRUE** if the SAMConnection object is successfully connected to a Messaging server or **FALSE** otherwise.

Examples

Example #23 - Checking whether there us a connection to a Messaging Server

```
<?php

if ($conn->isConnected()) {
 echo 'Connection is active.'
} else {
 echo 'No active connection!'
}

?>
```

See Also

- SAMConnection->disconnect()
- SAMConnection->connect()

SAMConnection->peek()

SAMConnection->peek() -- Read a message from a queue without removing it from the queue.

Description

SAMConnection

SAMMessage **peek** (string `$target` [, array `$properties`])

Parameters

`target`
: The identity of the queue from which to peek the message.

`properties`
: An optional associative array of properties describing other parameters to control the peek operation.

Property name	Possible values
SAM_CORRELID	This is the target correlation id string of the message. This would typically have been returned by a "send" request.
SAM_MESSAGEID	This is the message id string of the message which is to be peeked.

Return Values

This method returns a SAMMessage object or **FALSE** if an error occurs.

Examples

Example #24 - Retrieve the next message from a queue without removing it

```
<?php
$msg = $conn->peek('queue://receive/test');
```

```
if (!$msg) {
 // The peek failed!
 echo "Peek failed ($conn->errno) $conn->error";
}
?>
```

Example #25 - Retrieve a specific message from a queue without removing it from the queue

```
<?php

$msg = $conn->peek('queue://receive/test', array(SAM_MESSAGEID => $messageId));

?>
```

See Also

- SAMConnection->peekAll()

SAMConnection->peekAll()

SAMConnection->peekAll() -- Read one or more messages from a queue without removing it from the queue.

Description

SAMConnection

array **peekAll** (string `$target` [, array `$properties`])

Parameters

`target`
: The identity of the queue from which messages should be peeked.

`properties`
: An optional associative array of properties describing other parameters to control the peek operation.

Property name	Possible values
SAM_CORRELID	This is the target correlation id string of messages to be peeked. This would typically have been returned by a "send" request.
SAM_MESSAGEID	This is the message id string of a message which is to be peeked.

Return Values

This method returns an array of SAMMessage objects or **FALSE** if an error occurs.

Examples

Example #26 - Retrieve all messages in a queue without removing them

```
<?php
$msgArray = $conn->peekAll('queue://receive/test');
if ($msgArray) {
  foreach ( $msgArray as $key => $msg) {
      echo "Message $key: body = $msg->body\n";
  }
} else {
  echo "PeekAll failed ($conn->errno) $conn->error";
}
?>
```

Example #27 - Retrieve all messages from a queue with a matching correlation id

```
<?php

  $msgArray = $conn->peekAll('queue://receive/test', array(SAM_CORRELID =>
$correlId ));
  if ($msgArray) {

     foreach ( $msgArray as $key => $msg) {
           echo "Message $key: body = $msg->body\n";
        }
     }
  else
     echo "PeekAll failed ($conn->errno) $conn->error";
  }

?>
```

See Also

- SAMConnection->peek()

SAMConnection->receive()

SAMConnection->receive() -- Receive a message from a queue or subscription.

Description

SAMConnection

SAMMessage **receive** (string `$target` [, array `$properties`])

Parameters

`target`
The identity of the queue, topic or subscription from which to receive the message.

`properties`
An optional associative array of properties describing other parameters to control the receive operation.

Property name	Possible values
SAM_CORRELID	Used to request selection of the message to receive based upon the correlation id string of the message.
SAM_MESSAGEID	Used to request selection of the message to receive based upon the message id string of the message.
SAM_WAIT	Timeout value in milliseconds to control how long the request should block waiting to receive a message before returning with a failure if no message is available on the queue or topic. The default value is 0 meaning wait indefinitely and should be used with caution as the request may wait until the overall PHP script processing time limit has expired if no message becomes available.

Return Values

This method returns a SAMMessage object or **FALSE** if an error occurs.

Examples

Example #28 - Receiving a message from a queue

```
<?php
$msg = $conn->receive('queue://receive/test');

if (!$msg) {
 // The receive failed!
 echo "Receive failed ($conn->errno) $conn->error";
}
?>
```

Example #29 - Receiving a message from a queue with options

In this example the SAM_CORRELID option is used to specify a correlation id string to be used to identify the message to receive. A wait timeout of 10 seconds is also specified.

```
<?php

$msg = $conn->receive('queue://receive/test', array(SAM_CORRELID => $token,
SAM_WAIT => 10000));

?>
```

Example #30 - Receiving a message from a subscription

In this example we show how to receive a message from a subscription id.

```
<?php
$msg = $conn->receive($subscriptionName);

if (!$msg) {
 // The receive failed!
 echo "Receive failed ($conn->errno) $conn->error";
}
?>
```

Please note that $subscriptionName is a subscription id returned from an earlier subscribe call.

See Also

- SAMConnection->send()

SAMConnection->remove()

SAMConnection->remove() -- Remove a message from a queue.

Description

SAMConnection

SAMMessage **remove** (string `$target` [, array `$properties`])

Removes a message from a queue.

Parameters

`target`
: The identity of the queue from which to remove the message.

`properties`
: An optional associative array of properties describing other parameters to control the remove operation.

Property name	Possible values
SAM_CORRELID	This is the target correlation id string of the message. This would typically have been returned by a "send" request.
SAM_MESSAGEID	This is the message id string of the message which is to be removed.

Return Values

This method returns **FALSE** if an error occurs.

Examples

Example #31 - Removing a message from a queue by message id

```
<?php
if (!$conn->remove('queue://receive/test', array(SAM_MESSAGEID => $messageId)))
{
 // The remove failed!
 echo "Remove failed ($conn->errno) $conn->error";
}
?>
```

SAMConnection->rollback()

SAMConnection->rollback() -- Cancels (rolls back) an in-flight unit of work.

Description

SAMConnection

bool **rollback** (void)

Rolls back an in-flight unit of work.

Return Values

This method returns **FALSE** if an error occurs.

Examples

Example #32 - Cancelling an in-flight unit of work

```
<?php
if (!$conn->rollback()) {
 // The rollback failed!
 echo "Rollback failed ($conn->errno) $conn->error";
}
?>
```

See Also

- SAMConnection->commit()

SAMConnection->send()

SAMConnection->send() -- Send a message to a queue or publish an item to a topic.

Description

SAMConnection

string **send** (string `$target`, SAMMessage `$msg` [, array `$properties`])

The "send" method is used to send a message to a specific queue or to publish to a specific topic. The method returns a correlation id that can be used as a selector to identify reply or response messages when these are requested.

Parameters

`target`
: If sending a message, the identity of the queue (queue://queuename) or if publishing to a topic the identity of the topic (topic://topicname) to which the message is to be delivered.

`msg`
: The message to be sent or published.

`properties`
: An optional associative array of properties describing other parameters to control the receive operation.

Property name	Possible values
SAM_DELIVERYMODE	Indicates whether the messaging server should enusre delivery or whether it is acceptable for messages to be lost in the case of system failures. The value of this property may be set to either **SAM_PERSISTENT**, to indicate that message loss is not acceptable, or **SAM_NON_PERSISTENT**, if message loss is acceptable. The resulting behaviour of the send will vary depending on the capabilities of the messaging server the PHP script is currently connected to. If the server does not support persistent messages and **SAM_PERSISTENT** is specified the send request will fail with an error indication

	showing the capability is not available.
SAM_PRIORITY	A numeric value between 0 and 9 indicating the desired message delivery priority. A priority value of 0 indicates the lowest priority while 9 indicates highest priority. If no priority is specified a default will be assigned which is dependent on the messaging server being used.
SAM_CORRELID	A string to be assigned as a correlation id for this message. If no value is given the messaging server may assign a value automatically.
SAM_TIMETOLIVE	A time in milliseconds indicating how long the messaging server should retain the message on a queue before discarding it. The default value is 0 indicating the message should be retained indefinitely.
SAM_WMQ_TARGET_CLIENT	This property is only valid when using WebSphere MQ and indicates whether or not an RFH2 header should be included with the message. This option may be set to either 'jms' or 'mq'. The default is 'jms' which means that an RFH2 header is included. If the value 'mq' is specified then no RFH2 is included with the message.

Return Values

A correlation id string that can be used in a subsequent receive call as a selector to obtain any reply or response that has been requested or **FALSE** if an error occurred.

Note
A correlation id will only be returned for a successful send to a queue destination (queue://xxxx) in which case it will reflect the message identitiy of the message on the queue. In the case of a send being used to publish data to a topic the return value will be **TRUE** as no correlation id is availabe for return.

Examples

Example #33 - Send a message to a queue

```
<?php
$msg = new SAMMessage('This is a simple text message');
$correlId = $conn->send('queue://send/test', $msg);
if (!$correlId) {
   // The send failed!
   echo "Send failed ($conn->errno) $conn->error";
}

?>
```

Example #34 - Publish a message to a topic

```
<?php
$msg = new SAMMessage('This is a simple text item');
if (!$conn->send('topic://test', $msg)) {
   // The Send failed!
   echo "Send failed ($conn->errno) $conn->error";
}
?>
```

Example #35 - Send a request and receive a response

```
<?php
$msg = new SAMMessage('This is a simple text message');
$msg->header->SAM_REPLY_TO = 'queue://receive/test';
$correlid = $conn->send('queue://send/test', $msg);

if (!$correlid) {
   // The Send failed!
   echo "Send failed ($conn->errno) $conn->error";
} else {
   $resp = $conn->receive('queue://receive/test', array(SAM_CORRELID =>
$correlid));
}
?>
```

See Also

- SAMConnection->receive()

SAMConnection::setDebug()

SAMConnection::setDebug() -- Turn on or off additional debugging output.

Description

The "setdebug" method is used to turn on or off additional debugging output. The SAM framework will provide method/function entry and exit trace data plus additional information. Protocol specific implementations also provide extra output.

SAMConnection

void **send** (bool `$switch`)

Parameters

`switch`
: If this parameter is set to **TRUE** additional debugging output will be provided. If the value is set to **FALSE** output of additional information will be stopped.

Examples

Example #36 - Turn on debugging output

```
<?php
$conn->setdebug(TRUE);
?>
```

Example #37 - Turn off debugging output

```
<?php
$conn->setdebug(FALSE);
?>
```

SAMConnection->subscribe()

SAMConnection->subscribe() -- Create a subscription to a specified topic.

Description

SAMConnection

string **subscribe** (string `$targetTopic`)

The "subscribe" method is used to create a new subscription to a specified topic.

Parameters

`targetTopic`
The identity of the topic (topic://topicname) to subscribe to.

Return Values

A subscription identifier that can be used in a subsequent receive call as a selector to obtain any topic data or **FALSE** if an error occurred. The subscription identifier should be used in the receive call in place of the simple topic name.

Examples

Example #38 - Subscribe to a topic

```
<?php
$subid = $conn->subscribe('topic://A');
if (!$subid) {
 // The subscribe failed!
 echo "Subscribe failed ($conn->errno) $conn->error";
}
?>
```

See Also

- SAMConnection->unsubscribe()

SAMConnection->unsubscribe()

SAMConnection->unsubscribe() -- Cancel a subscription to a specified topic.

Description

SAMConnection

bool **unsubscribe** (string `$subscriptionId` [, string `$targetTopic`])

The "unsubscribe" method is used to delete an existing subscription to a specified topic.

Parameters

`subscriptionId`
: The identifier of an existing subscription as returned by a call to the subscribe method.

Return Values

This method returns **FALSE** if an error occurs.

Examples

Example #39 - Delete a subscription

```
<?php
if (!$conn->unsubscribe($subid)) {
   // The unsubscribe failed!
   echo "Unsubscribe failed ($conn->errno) $conn->error";
}
?>
```

See Also

- SAMConnection->subscribe()

SAMMessage->body

SAMMessage->body -- The body of the message.

Description

SAMMessage

string *nody*;

The "body" property contains the actual body of the message. It may not always be set.

Examples

Example #40 - Setting a text string into the body of a message

```
<?php
$msg = new SAMMessage();
$msg->body = 'This is a simple message';

?>
```

See Also

- SAMMessage->header

SAMMessage->__construct()

SAMMessage->__construct() -- Creates a new Message object

Description

SAMMessage

__construct ([mixed `$body`])

Creates a new SAMMessage object optionally specifying a message body.

Parameters

body
The optional body for the message.

Examples

Example #41 - Creating a message

```
<?php

$msg = new SAMMessage();

?>
```

Example #42 - Creating a message with a simple text payload

```
<?php

$msg = new SAMMessage('This is a simple text message');

?>
```

SAMMessage->header

SAMMessage->header -- The header properties of the message.

Description

SAMMessage

object *header*;

The *header* property is a container for any system or user properties that area associated with the message.

Properties may be assigned by the sender of a message to control the way the messaging systems handles it or may be assigned by the messaging system itself to tell the recipient extra information about the message or the way in which it has been handled.

Some properties are understood by SAM in which case constants have been defined for them. The majority of properties however are ignored by the SAM implementation and simply passed through to the underlying messaging systems allowing the application to use messaging specific property names or to define its own "user" properties.

The SAM defined properties are as follows:

Property name	**Possible values**
SAM_MESSAGEID	When a message is received this field contains the unique identifier of the message as allocated by the underlying messaging system. When sending a message this field is ignored.
SAM_REPLY_TO	A string providing the identity of the queue on to which responses to this message should be posted.
SAM_TYPE	An indication of the type of message to be sent. The value may be **SAM_TEXT** indicating the contents of the message body is a text string, or **SAM_BYTES** indicating the contents of the message body are some application defined format. The way in which this property is used may

	depend on the underlying messaging server. For instance a messaging server that supports the JMS (Java Message Service) specification may interpret this value and send messages of type "jms_text" and "jms_bytes". In addition, if the SAM_TYPE property is set to **SAM_TEXT** the data provided for the message body is expected to be a UTF8 encoded string.

When setting the values of properties it is often useful to give a hint as to the format in which the property should be delivered to the messaging system. By default property values are delivered as text and the following simple syntax may be used to set a value:

Example #43 - Setting a text format property using the default syntax

```
<?php
$msg = new SAMMessage();

$msg->header->myPropertyName = 'textData';
?>
```

If it is desired to pass type information an alternative syntax may be used where the value and the type hint are passed in an associative array:

Example #44 - Setting a text format property using a type hint

```
<?php
$msg = new SAMMessage();

$msg->header->myPropertyName = array('textData', SAM_STRING);
?>
```

When passing a type hint the type entry should be one of the SAM defined constant values as defined by the following table:

Constant	Type description
SAM_BOOLEAN	Any value passed will be interpreted as logical true or false. If the value cannot be interpreted as a PHP boolean value the value passed to the messaging system is

	undefined.
SAM_BYTE	An 8-bit signed integer value. SAM will attempt to convert the property value specified into a single byte value to pass to the messaging system. If a string value is passed an attempt will be made to interpret the string as a numeric value. If the numeric value cannot be expressed as an 8-bit signed binary value data may be lost in the conversion.
SAM_DOUBLE	A long floating point value. SAM will attempt to convert the property value specified into a floating point value with 15 digits of precision. If a string value is passed an attempt will be made to interpret the string as a numeric value. If the passed value cannot be expressed as a 15 digit floating point value data may be lost in the conversion.
SAM_FLOAT	A short floating point value. SAM will attempt to convert the property value specified into a floating point value with 7 digits of precision. If a string value is passed an attempt will be made to interpret the string as a numeric value. If the passed value cannot be expressed as a 7 digit floating point value data may be lost in the conversion.
SAM_INT	An 32-bit signed integer value. SAM will attempt to convert the property value specified into a 32-bit value to pass to the messaging system. If a string value is passed an attempt will be made to interpret the string as a numeric value. If the numeric value cannot be expressed as an 32-bit signed binary value data may be lost in the conversion.
SAM_LONG	An 64-bit signed integer value. SAM will attempt to convert the property value specified into a 64-bit value to pass to the messaging system. If a string value is passed an attempt will be made to interpret the string as a numeric value. If the numeric value cannot be expressed as an 64-bit signed binary value data may be lost in the conversion.

SAM_STRING	SAM will interpret the property value specified as a string and pass it to the messaging system accordingly.

Examples

Example #45 - Setting properties as the sender of a message

```
<?php
$msg = new SAMMessage('This is a test message');

// defining SAM specific properties...
$msg->header->SAM_REPLY_TO = 'queue://test/replyQueue';

// defining arbitrary properties...
//
// a default string property
$msg->header->myStringProp1 = 'a string property';
// a string property with a type hint
$msg->header->myStringProp2 = array('another string property', SAM_STRING);

// a boolean property
$msg->header->myBoolProp = array(FALSE, SAM_BOOL);

// numeric format properties
$msg->header->myIntProp = array(32768, SAM_INT);
$msg->header->myLongProp = array(9876543, SAM_LONG);
$msg->header->myByteProp1 = array(123, SAM_BYTE);
$msg->header->myByteProp2 = array('12', SAM_BYTE);
$msg->header->myFloatProp = array(3.141592, SAM_FLOAT);
$msg->header->myDoubleProp = array(3.14159265358979, SAM_DOUBLE);
?>
```

Example #46 - Retreiving property values from a message

```
<?php

// accessing an application specific property
$intProp = $msg->header->'MyIntProp';

// accessing a messaging system specific property
$encoding = $msg->header->'JMS_IBM_Msgtype';

?>
```

See Also

- SAMMessage->body