QAnywhere™

Version 12.0.1

January 2012

Version 12.0.1
January 2012

Contents

About this book

This book describes QAnywhere, which is a messaging platform for mobile, wireless, desktop, and laptop clients.

QAnywhere technology

QAnywhere helps you develop application-to-application messaging for mobile devices. Application-to-application messaging permits communication between custom programs running on mobile or wireless devices and a centrally-located server application.

Using store-and-forward technology, QAnywhere provides secure, assured message delivery for remote and mobile applications. Because QAnywhere automatically handles the challenges of slow and unreliable networks, you can concentrate on application functionality instead of issues surrounding connectivity, communication, and security. QAnywhere store-and-forward technology ensures that your applications are always available, even when a network connection is not.

QAnywhere provides communication in occasionally-connected environments. It handles the challenges of wireless networks, such as slow speed, spotty geographic coverage, and dropped network connections. The store-and-forward nature of QAnywhere messaging means that messages can be constructed even when the destination application is not reachable over the network; the message is delivered when the network becomes available.

QAnywhere messages are exchanged via a central server, so that the sender and recipient of a message never have to be connected to the network at the same time.

QAnywhere has the following additional features:

- QAnywhere can protect proprietary or sensitive information by encrypting all messages sent over public networks.
- You can customize the delivery of messages using transmission rules so that, for example, large low-priority messages are transmitted during off-peak hours.
- QAnywhere messages can be transported over TCP/IP, HTTP, or HTTPS protocols. They can also be delivered from a Windows Mobile handheld device by Microsoft ActiveSync. The message itself is independent of the network protocol, and can be received by an application that communicates over a different network.
- QAnywhere compresses data sent between mobile applications and enterprise servers.
- QAnywhere provides a C++, Java, .NET, and SQL API to provide solutions to developers with different skill sets.
- QAnywhere permits seamless communication with other messaging systems that have a JMS interface. This allows integration with Java EE applications.
- QAnywhere includes a mobile web services interface that helps you create reliable mobile applications based on enterprise web services.

QAnywhere is built on MobiLink synchronization technology.

What QAnywhere does

QAnywhere provides the following application-to-application features and components.

- **QAnywhere API** The object-oriented QAnywhere API provides the infrastructure to build messaging applications for Windows desktop and Windows Mobile devices. The QAnywhere API is available in Java, C++, .NET, and SQL.
- **Store-and-forward** QAnywhere applications store messages locally until a connection between the client and the server is available for data transmission.
- **Complements data synchronization** QAnywhere applications use relational databases as a temporary message store. The relational database ensures that the message store has security, transaction-based computing, and the other benefits of relational databases.

  The use of SQL Anywhere relational databases as message stores makes it easy to use QAnywhere together with a data synchronization solution. Both use MobiLink synchronization as the underlying mechanism for exchanging information between client and server.
- **Integration with external messaging systems** In addition to exchanging messages among QAnywhere applications, you can integrate QAnywhere clients into external messaging systems that support a JMS interface.
- **Encryption** Messages can be sent encrypted using transport-layer security and can be encrypted using simple encryption or any FIPS-certified AES algorithm.
- **Compression** Message content can be stored compressed using the popular ZLIB compression library.
- **Authentication** You can authenticate QAnywhere clients using a built-in facility or through custom authentication scripts (including existing authentication services used in your organization).
- **Multiple networks** QAnywhere works over any wired or wireless network that supports TCP/IP or HTTP.
- **Failover** You can run multiple MobiLink servers so that there are alternate servers if one fails.
- **Administration** A QAnywhere application can browse and manipulate messages on the client and server side.
- **Multiple queues** Support for multiple arbitrarily-named queues on client devices permits multiple client applications to coexist on a single device. Applications can send and receive on any number of queues. Messages can be sent between applications that are coexisting on the same device and between applications on different devices.
- **Server-initiated send and receive** QAnywhere can push messages to client devices, allowing client applications to implement message-driven logic.
- **Transmission rules** You can create rules that specify when message transmission should occur.

- **Resumable downloads** Large messages or groups of messages are sent to QAnywhere clients in piecemeal fashion to minimize the retransmission of data during network failures.
- **Guaranteed delivery** QAnywhere guarantees the delivery of messages.
- **Mobile web services** Mobile web services help the transport of web service requests and responses over QAnywhere.

QAnywhere architecture

This section explains the architecture of QAnywhere messaging applications. The discussion begins with a simple messaging scenario and then progresses to more advanced scenarios.

Client applications send and receive messages using the QAnywhere API. Messages are queued in the client message store. Message transmission is the exchange of messages between client message stores through a central QAnywhere server message store.

The following typical messaging scenarios are supported by QAnywhere:

- **Local application-to-application messaging** For exchanging messages between applications using a SQL Anywhere database as the local message store. Messages are transmitted between applications via their connection to the database.
- **Simple client/server messaging** For exchanging messages among QAnywhere clients. Client applications control when to transmit messages between the client and server message stores.

  See “Simple messaging scenario” on page 5.
- **Client/server messaging with push notifications** For exchanging messages among QAnywhere clients. In this scenario, the MobiLink server can initiate message transmission between clients. This is done by exchanging messages between client and server message stores.
- **Client/server messaging with external messaging systems** For exchanging messages among QAnywhere clients over an external system that supplies a JMS provider, such as BEA WebLogic or Sybase EAServer.

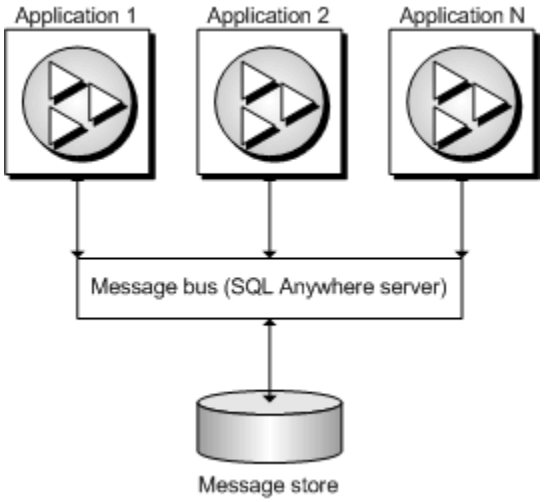
Push notifications and external messaging systems can be used together, providing the most general solution.

See also

- “Setting up the local message store” on page 17
- “Simple messaging scenario” on page 5
- “Scenario for messaging with push notifications” on page 6
- “Scenario for messaging with external messaging systems” on page 7

Application-to-application messaging scenario

Application-to-application messaging consists of applications transmitting messages via a SQL Anywhere database acting as a local message store. The messages are transmitted between applications through their connection to the database. The following diagram illustrates a typical scenario where two applications use a local message store to transmit messages between them.



This setup includes the following components:

- **Local message store** Messages are stored in the SQL Anywhere database. The database must be set up as a local message store.
- **QAnywhere Agent** The QAnywhere Agent manages the transmission of messages. This process is independent of QAnywhere applications.
- **QAnywhere application** An application written using the QAnywhere C++, Java, or .NET API makes method calls to send and receive messages. The basic object used by the client application is the QAManager.

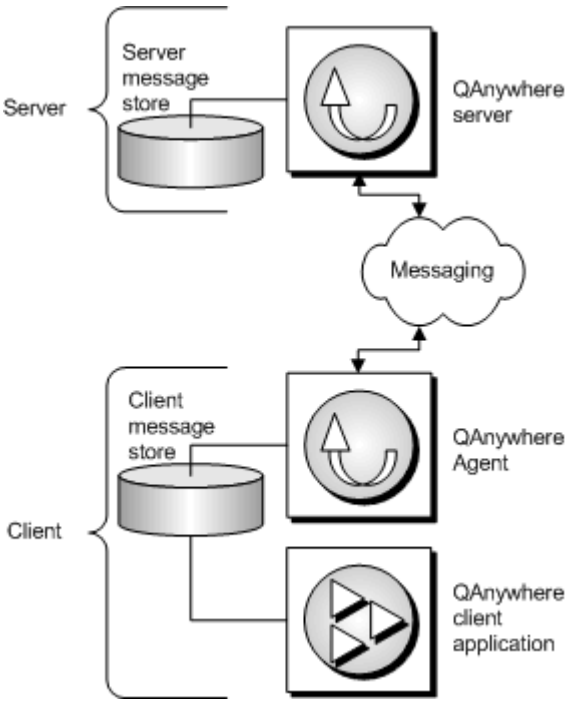
Messages are sent and received by QAnywhere applications connected to the SQL Anywhere database. Messages are not picked up until an application initiates a message transmission. QAnywhere applications use **policies** to determine when to perform a message transmission. Policies include on-demand, automatic, scheduled, and custom. The on-demand policy permits the user to control message transmission. The automatic policy initiates a message transmission whenever a message to or from the client is ready for delivery. The custom policy uses transmission rules to add further control over message transmission.

See also

- "Starting the QAnywhere agent" on page 44
- "How to write QAnywhere client applications" on page 49
- "Determining when message transmission should occur on the client" on page 46

Simple messaging scenario

A simple QAnywhere messaging setup is illustrated in the following diagram. For simplicity, only a single client is shown. However, a typical scenario has multiple clients with the server message store existing to transmit messages between them.



This setup includes the following components:

- **Server message store** At the server, the messages are stored in a relational database. The database must be set up as a MobiLink consolidated database, and may be any supported consolidated database.
- **Client message store** The messages at each client are stored in a relational database. QAnywhere supports SQL Anywhere and UltraLite databases. SQL Anywhere databases are recommended for data synchronization applications. UltraLite databases are recommended for applications used exclusively for storing and forwarding messages.
- **QAnywhere server** The QAnywhere server is a MobiLink server that is enabled for messaging. MobiLink synchronization provides the transport for transmitting and tracking messages between QAnywhere clients and the server. MobiLink provides security, authentication, encryption, and flexibility. It also allows messaging to be combined with data synchronization.

  To start the QAnywhere server, start the MobiLink server with the -m option.
- **QAnywhere Agent** The QAnywhere Agent manages the transmission of messages on the client side. This process is independent of QAnywhere client applications.

- **QAnywhere client application** An application written using the QAnywhere C++, Java, or .NET API makes method calls to send and receive messages. The basic object used by the client application is the QAManager.

Messages are sent and received by QAnywhere clients. Messages at the server are not picked up until the client initiates a message transmission. QAnywhere clients use **policies** to determine when to perform a message transmission. Policies include on demand, automatic, scheduled, and custom. The on demand policy permits the user to control message transmission. The automatic policy initiates a message transmission whenever a message to or from the client is ready for delivery. The custom policy uses transmission rules to add further control over message transmission.

See also

- "Starting QAnywhere with MobiLink enabled" on page 29
- "How to write QAnywhere client applications" on page 49
- "Starting the QAnywhere agent" on page 44
- "Determining when message transmission should occur on the client" on page 46

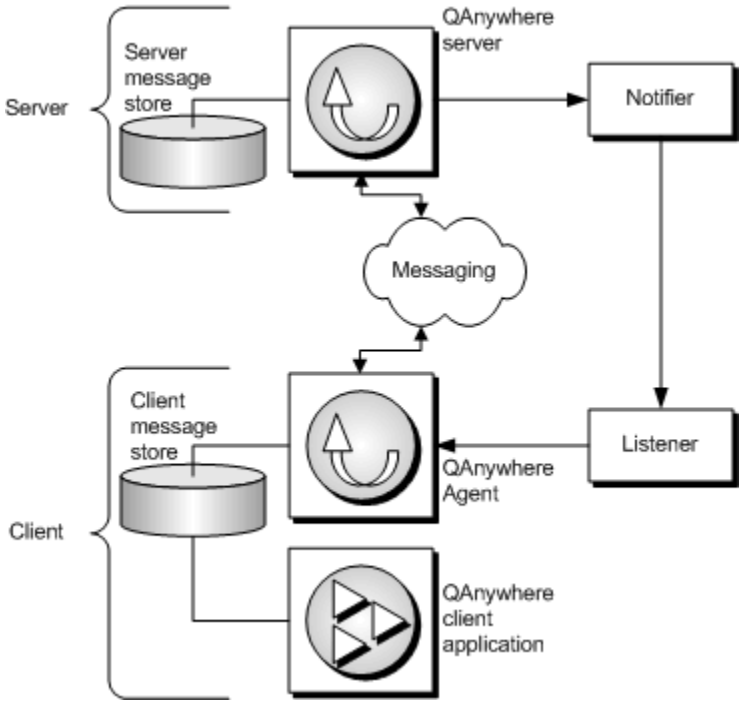
Scenario for messaging with push notifications

A push notification is a special message delivered from the server to a QAnywhere client. The push notification occurs when a message arrives at the server message store. The messaging server automatically notifies the recipient client Listener of the push request. The client initiates message transmission to receive messages waiting at the server or takes a custom action.

Push notifications introduce two extra components to the QAnywhere architecture. At the server, a QAnywhere Notifier sends push notifications. At the client, a QAnywhere Listener receives these push notifications and passes them on to the QAnywhere Agent.

If you do not use push notifications, messages are still transmitted from the server message store to the client message store, but the transmission must be initiated at the client, such as by using a scheduled transmission policy.

The architecture for messaging with push notifications is an extension of that described in "Simple messaging scenario" on page 5. The following diagram shows this architecture:



The following components are added to the simple messaging scenario to enable push notification:

- **QAnywhere Notifier** The Notifier is the component of the MobiLink server that is used to deliver push notifications.

  The QAnywhere Notifier is a specially configured instance of the Notifier that sends push notifications when a message is ready for delivery.

- **Listener** The Listener is a separate process that runs at the client. It receives push notifications and passes them on to the QAnywhere Agent. QAnywhere Agent policies determine if push notifications automatically cause message transmission.

See also

- “Determining when message transmission should occur on the client” on page 46
- “Push notifications” on page 32
- “Receiving messages asynchronously” on page 69
- “Server-initiated synchronization” [*MobiLink - Server-Initiated Synchronization*]

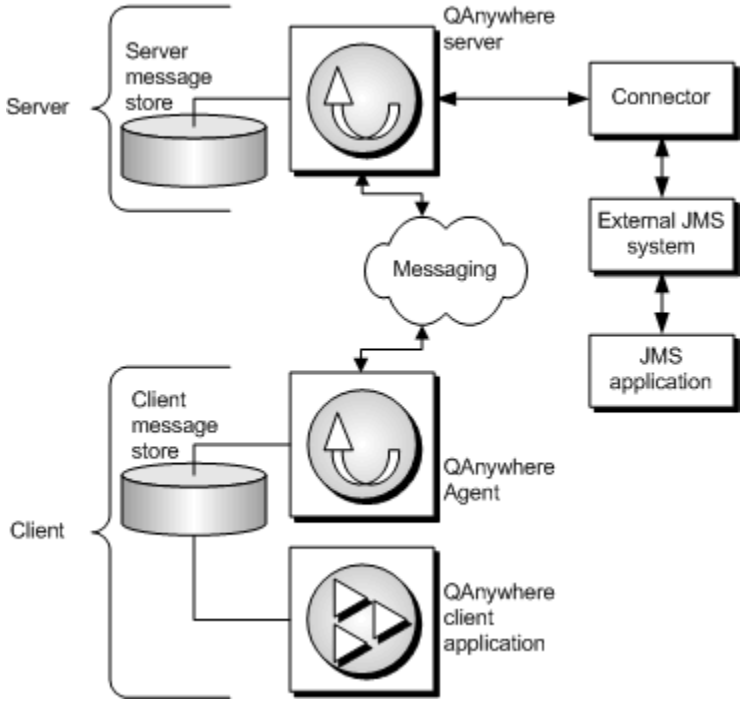
Scenario for messaging with external messaging systems

In addition to exchanging messages among QAnywhere applications, you can exchange messages with systems that have a JMS interface using a specially configured client known as a connector. JMS is the Java Message Service API for adding messaging capabilities to Java applications.

The external messaging system is set up to act like a special client. It has its own address and configuration.

The architecture for messaging with external messaging systems is an extension of the architecture described in "Simple messaging scenario" on page 5.

The following diagram shows this architecture:



The component that is added to the simple messaging scenario to enable messaging with an external messaging system is as follows:

- **QAnywhere JMS Connector** The JMS Connector provides an interface between QAnywhere and the external messaging system.

  The JMS Connector is a special QAnywhere client that moves messages between QAnywhere and the external JMS system.

See also

- "Connectors" on page 129
- "Tutorial: Using JMS connectors" on page 140

QAnywhere message delivery

Messages are sent from a client message store to a server message store, and then on to another client message store. QAnywhere does this via queues: a message is put on a queue in the client message store;

when it is received by the server message store, it is put on a queue for delivery to one or more client message stores; and when it is received by a client message store, it is put on a queue for pickup.

Once a message is sent, it gets delivered unless one of the following occurs:

- The message expires (only if an expiration is specified).
- The message is canceled via Sybase Central or via the cancelMessage API call.
- The device from which the message is sent is lost irretrievably before it can synchronize with the server message store (or for some other reason, synchronization is impossible).

A message does not get delivered more than once. If an application successfully acknowledges or commits the receipt of a message, then the same message is not delivered again. There is a possible exception with JMS servers: in the event of the MobiLink server or JMS server crashing, there is a possibility that a message could get delivered twice.

Deciding between SQL Anywhere and UltraLite

QAnywhere client applications can now use an UltraLite database as the client message store. This provides a lighter-weight solution for pure messaging applications on mobile devices. A pure messaging application means an application that uses store and forward messaging, but not data synchronization.

Some of the key advantages of UltraLite are:

- It has a smaller application footprint and does not require full SQL Anywhere installation.
- It has a smaller process footprint. QAnywhere Agent requires only 3 processes instead of 4 (uleng12, dblsn, and qauagent instead of dbeng12, dbmlsync, dblsn, and qaagent).

UltraLite limitations

Keep in mind the following limitations of UltraLite when deciding between SQL Anywhere and UltraLite:

- There is no support for "ESCAPE" keyword in transmission rule condition syntax.
- There is limited support for property attributes. You can only use property attribute functionality with the predefined property ias_Network.

Recommendation

UltraLite should always be used instead of SQL Anywhere when SQL Anywhere is not already installed. SQL Anywhere is available for circumstances where you want to add messaging along side an already implemented SQL Anywhere data synchronization solution. However, UltraLite is recommended in all pure messaging environments.

See also

- “Custom client message store properties” on page 718

QAnywhere 12 plug-in

The Sybase Central QAnywhere 12 plug-in helps you create and administer your QAnywhere application. With the plug-in, you can:

- Create client and server message stores.
- Create and maintain configuration files for the QAnywhere Agent.
- Browse QAnywhere Agent log files.
- Create or modify destination aliases.
- Create JMS connectors and web service connectors.
- Create and maintain transmission rules files.
- Browse message stores remotely.
- Track messages.

Start the QAnywhere 12 plug-in

1. Start Sybase Central:

   Click **Start** » **Programs** » **SQL Anywhere 12** » **Administration Tools** » **Sybase Central**.

2. Click **Connections** » **Connect With QAnywhere 12**.

3. Specify an **ODBC Data Source Name** or **ODBC Data Source File**, and the **User ID** and **Password** if required.

4. Click **Connect**.

Quick start to QAnywhere

Set up and run QAnywhere messaging

1. Set up a local application-to-application message store.

   See “Setting up the local message store” on page 17.

   OR

   Set up a server message store or use an existing MobiLink consolidated database.

   See “Server message store setup” on page 22.

2. Start the MobiLink server with the -m option and a connection to the server message store.

See "Starting QAnywhere with MobiLink enabled" on page 29.

3. Set up client message stores. These are SQL Anywhere or UltraLite databases that are used to temporarily store messages.

   See "Setting up the client message store" on page 23.

4. For each client, write a messaging application.

   See "How to write QAnywhere client applications" on page 49.

5. If you want to integrate with an external JMS messaging system, set up JMS messaging for QAnywhere.

   See "Connectors" on page 129.

6. For each client, start the QAnywhere Agent with a connection to the local client message store.

   See "Starting the QAnywhere agent" on page 44.

For information about setting up mobile web services, see "Mobile web services" on page 89.

Other resources for getting started

- "Tutorial: Exploring TestMessage" on page 171
- "Tutorial: Using JMS connectors" on page 140
- Sample applications are installed to *%SQLANYSAMP12%\QAnywhere*.
- You can post questions on the SQL Anywhere Forum: http://sqlanywhere-forum.sybase.com

QAnywhere messages

QAnywhere messages consist of the following parts:

- headers
- properties
- content

Message properties can be referenced in transmission rules and delete rules or in your application.

The following sections describe message headers and properties, and how you can set them in QAnywhere messages.

Notes

- Message headers, message properties, and message content cannot be altered after the message is sent.
- You can read message headers, message properties, and message content after a message is received. If you are using the QAnywhere SQL API, these become unreadable after a commit or rollback occurs.
- The content is unreadable after acknowledgement or commit in all APIs.

Message headers

All QAnywhere messages support the same set of header fields. Header fields contain values that are used by both clients and providers to identify and route messages. How you use the headers depends on the type of client application you have.

QAnywhere supports the following predefined message headers:

- Message ID
- Message creation timestamp
- Reply-to address
- Message address
- Redelivered state of message
- Expiration of message
- Priority of message
- Message ID of a message for which this message is a reply

See also

- “Message headers” on page 656

Message properties

Each message contains a built-in facility for supporting application-defined property values. These message properties allow you to implement application-defined message filtering.

Message properties are name-value pairs that you can optionally insert into messages to provide structure. For example, in the .NET API the predefined message property ias_Originator, identified by the constant MessageProperties.ORIGINATOR, provides the message store ID that sent the message. Message properties can be used in transmission rules to determine the suitability of a message for transmission.

There are two types of message property:

- **Predefined message properties** These message properties are always prefixed with ias_ or IAS_.
- **Custom message properties** These are message properties that you defined. You cannot prefix them with ias_ or IAS_.

In either case, you access message store properties using get and set methods and pass the name of the predefined or custom property as the first parameter. See "Message property management" on page 661.

Predefined message properties

Some message properties have been predefined for your convenience. Predefined properties can be read but should not be set. The predefined message properties are:

- ias_Adapters
- ias_DeliveryCount
- ias_MessageType
- ias_RASNames
- ias_NetworkStatus
- ias_Originator
- ias_Status
- ias_StatusTime

For details about message properties, see "Message properties" on page 658.

See also

- "Predefined message properties" on page 659
- "Custom message properties" on page 661

Understanding destinations

With QAnywhere, messages are addressed to a destination. A destination always consists of an identifier and a queue name, separated by a backslash (\). For example:

```
ianywhere.connector.tibco\SomeQueue
 DEV007\app_queue1
 SalesTeam\queue1
```

The meaning of the identifier before the backslash depends on whether the message is addressed to a JMS application, destination alias, or a mobile application.

The first example illustrates the case where the message is addressed to a JMS application. In this case the identifier is the ID of a JMS connector running in the MobiLink server.

The second example illustrates the case where the message is addressed to a mobile application. In this case, the identifier is a message store ID of a QAnywhere message store.

The third example illustrates the case where the message is addressed to a destination alias. In this case, the identifier is a destination alias name.

The queue name in a destination refers to a queue defined in the JMS system when the identifier is a JMS connector ID, and refers to a QAnywhere application queue when the identifier is either a message store ID or a destination alias.

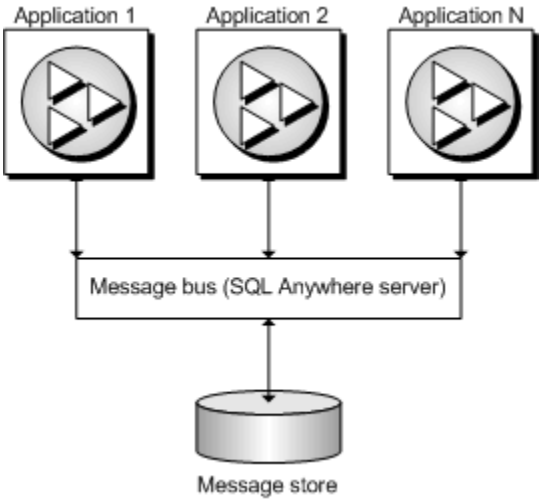
Note
QAnywhere destinations should always be specified using EN characters.

See also

- "JMS connectors" on page 129
- "Setting up the client message store" on page 23
- "-id qaagent option" on page 678
- "Destination aliases" on page 127
- "Sending QAnywhere messages" on page 61
- "Determining when message transmission should occur on the client" on page 46

QAnywhere message stores

A QAnywhere message store is a repository where message are temporarily stored. Messages can be temporarily stored on the server in a **server message store**, on the client in a **client message store**, or on the server in an **archive message store**, where they can be saved for a specified period of time.



See also

- "Server message stores" on page 21
- "Client message stores" on page 23

Local message stores

In a local message store configuration, applications are connected to a SQL Anywhere database acting as a local message bus and messages are transmitted between applications via their connection to the database.

Setting up the local message store

The local message store requires that you install QAnywhere. QAnywhere is located under the Synchronization and Messaging feature in the SQL Anywhere install program. In order to enable messaging in your SQL Anywhere database, you must install the QAnywhere schema into your SQL Anywhere database so that it can be used as a local message store. This is accomplished using the **-sil** option of the QAnywhere Agent for SQL Anywhere. The **-sil** option instructs the Agent to initialize the database as a local message store. The agent is simply used to initialize a local message store and is not needed any further. All QAnywhere objects created in the database belong to the **ml_qa_message_group** owner. Once your SQL Anywhere database has been initialized as a local message store, applications can use the QAnywhere client API to exchange messages.

Create a local message store (.NET example)

1. Create a SQL Anywhere database. You can omit this step if you are going to use an existing SQL Anywhere database.

```
dbinit localmsgstore.db
```

2. Install the QAnywhere schema into the SQL Anywhere database so that it can be used as a local message bus:

```
qaagent -sil -c "dbf=localmsgstore.db;uid=dba;pwd=sql"
```

3. Create a sender application to put messages in the message store. For information about using the .NET version of the QAnywhere client API, see "QAnywhere .NET API reference for clients" on page 181.

```
using System;
using System.IO;
using iAnywhere.QAnywhere.Client;
namespace sender
{
class sender
    {
        public static void Main() {
            try {
                // QAnywhere initialization
                QAManager mgr =
QAManagerFactory.Instance.CreateQAManager();
                // Be sure to set the DATABASE_TYPE property
                mgr.SetProperty( "DATABASE_TYPE", "sqlanywhere" );
                mgr.SetProperty( "CONNECT_PARAMS",
"dbf=localmsgstore.db;uid=dba;pwd=sql" );
                mgr.Open( AcknowledgementMode.IMPLICIT_ACKNOWLEDGEMENT );
                mgr.Start();
                // Create a text message
                QATextMessage msg = mgr.CreateTextMessage();
                msg.Text = "Sample text";
                // Queue the message
                mgr.PutMessage( "dbqueue", msg );
                // QAnywhere finalization
                mgr.Stop();
                mgr.Close();
                } catch( Exception exc ) {
                Console.WriteLine( exc.Message );
            }
        }
    }
}
```

4. Compile the program using the following command line. You must have Visual Studio installed on your computer.

```
csc /reference:"%SQLANY12%\Assembly\v2\iAnywhere.QAnywhere.Client.dll"
sender.cs
```

5. Create a receiver application to retrieve messages from the message store. For information about using the .NET version of the QAnywhere client API, see "QAnywhere .NET API reference for clients" on page 181.


```
using System;
using System.IO;
using iAnywhere.QAnywhere.Client;
namespace receiver
{
class receiver
{
public static void Main() {
try {
// QAnywhere initialization
QAManager mgr = QAManagerFactory.Instance.CreateQAManager();
// Be sure to set the DATABASE_TYPE property
mgr.SetProperty( "DATABASE_TYPE", "sqlanywhere" );
mgr.SetProperty( "CONNECT_PARAMS",
"dbf=localmsgstore.db;uid=dba;pwd=sql" );
mgr.Open( AcknowledgementMode.IMPLICIT_ACKNOWLEDGEMENT );
mgr.Start();
// Get the message
QATextMessage msg = (QATextMessage)mgr.GetMessage( "dbqueue" );
// Display the text
Console.WriteLine( msg.Text );
// QAnywhere finalization
mgr.Stop();
mgr.Close();
} catch( Exception exc ) {
Console.WriteLine( exc.Message );
}
}
}
}
```

6. Compile the program using the following command line. You must have Visual Studio installed on your computer.

   ```
   csc /reference:"%SQLANY12%\Assembly\v2\iAnywhere.QAnywhere.Client.dll"
   receiver.cs
   ```

7. Start the SQL Anywhere database:

   ```
   dbsrv12 localmsgstore.db
   ```

8. Run the sender application:

   ```
   sender
   ```

9. Run the receiver application:

   ```
   receiver
   ```

   The string *"Sample Text"* is displayed.

Create a local message store (Java example)

1. Create a SQL Anywhere database. You can omit this step if you are going to use an existing SQL Anywhere database.

   ```
   dbinit localmsgstore.db
   ```

2. Install the QAnywhere schema into the SQL Anywhere database so that it can be used as a message bus.

```
qaagent -sil -c "dbf=localmsgstore.db;uid=dba;pwd=sql"
```

3. Create a sender application to put messages in the message store. For information about using the .NET version of the QAnywhere client API, see "QAnywhere Java API reference for clients" on page 468.

```
import java.util.*;
import ianywhere.qanywhere.client.*;
public class sender
{
public static void main( String [] args ) {
try {
// QAnywhere initialization
QAManager mgr = QAManagerFactory.getInstance().createQAManager();
// Be sure to specify the DATABASE_TYPE
mgr.setProperty( "DATABASE_TYPE", "sqlanywhere" );
mgr.setProperty( "CONNECT_PARAMS",
"dbf=localmsgstore.db;uid=dba;pwd=sql" );
mgr.open( AcknowledgementMode.IMPLICIT_ACKNOWLEDGEMENT );
mgr.start();
// Create a text message
QATextMessage msg = mgr.createTextMessage();
msg.setText( "Sample text" );
// Queue the message
mgr.putMessage( "dbqueue", msg );
// QAnywhere finalization
mgr.stop();
mgr.close();
} catch( Exception exc ) {
System.out.println( exc.getMessage() );
}
}
}
```

4. Compile the program using the following command line. You must have the Java JDK installed on your computer.

```
javac  -cp "%SQLANY12%\java\qaclient.jar" sender.java
```

5. Create a receiver application to retrieve messages from the message store. For information about using the Java version of the QAnywhere client API, see "QAnywhere Java API reference for clients" on page 468.

```
import java.util.*;
import ianywhere.qanywhere.client.*;
public class receiver
{
public static void main( Strings [] args ) {
try {
// QAnywhere initialization
QAManager mgr = QAManagerFactory.getInstance().createQAManager();
// Be sure to set the DATABASE_TYPE property.
mgr.setProperty( "DATABASE_TYPE", "sqlanywhere" );
mgr.setProperty( "CONNECT_PARAMS",
"dbf=localmsgstore.db;uid=dba;pwd=sql" );
mgr.open( AcknowledgementMode.IMPLICIT_ACKNOWLEDGEMENT );
mgr.start();
// Get the message
QATextMessage msg = (QATextMessage)mgr.getMessage( "dbqueue" );
// Display the text
System.out.println( msg.getText() );
```



```
// QAnywhere finalization
mgr.stop();
mgr.close();
} catch( Exception exc ) {
System.out.println( exc.getMessage() );
}
}
}
```

6. Compile the program using the following command line. You must have the Java JDK installed on your computer.

   ```
   javac -cp "%SQLANY12%\java\qaclient.jar" receiver.java
   ```

7. Start the SQL Anywhere database:

   ```
   dbsrv12 localmsgstore.db
   ```

8. Run the sender application:

   ```
   java -cp ".\;%SQLANY12%\java\qaclient.jar;%SQLANY12%\java\jodbc.jar"
   sender
   ```

9. Run the receiver application:

   ```
   java -cp ".\;%SQLANY12%\java\qaclient.jar;%SQLANY12%\java\jodbc.jar"
   receiver
   ```

   The string *"Sample Text"* is displayed.

Cleaning the message store

By default, messages that have reached a final state remain in the message store. After a period of use, messages accumulate causing your database to grow. When you initialized your database for use as a message bus, a stored procedure called **ml_qa_clearreceivedmessages** was created. This stored procedure will delete messages that have reached a final state from the message store. A convenient way to manage the growth of messages in your database is to create a database event that, when triggered, invokes the **ml_qa_clearreceivedmesssages** stored procedure. For example, the following database event causes messages in a final state to be deleted every day at midnight:

```
CREATE EVENT message_cleanup
SCHEDULE
START TIME '12:00AM' EVERY 24 HOURS
HANDLER begin call ml_qa_message_group.ml_qa_clearreceivedmessages() end
```

Viewing messages

Use Sybase Central to view messages in your SQL Anywhere database by connecting to a client store using the QAnywhere plug-in.

Server message stores

The server message store is a relational database on the server that temporarily stores messages until they are transmitted to a client message store, web service or JMS system. Messages are exchanged between clients via the server message store.

A server message store is a MobiLink consolidated database and can be any RDBMS that MobiLink supports, with the exception of MySQL. You can create a new database for this purpose, or use an existing database.

Server message store setup

When setting up the server message store, an archive message store is created. The archive message store is a set of tables that coexist with the server message store, and stores all messages waiting to be deleted. A regularly executed system process transports messages between the message stores by removing all messages in the server message store that have reached a final state and then inserting them into the archive message store. Messages remain in the archive message store until deleted by a server delete rule. Usage of the archive message store improves the performance of the server message store by minimizing the amount of messages that need to be filtered during synchronization. See "Archive message store requests" on page 149.

To set up a database to use as a server message store, you run a setup script. The consolidated database should be configured for case insensitive comparisons and string operations. If you use the **Create Synchronization Model Wizard** to create your consolidated database, the setup is done for you.

See "Consolidated database setup" [*MobiLink - Server Administration*].

For information about creating SQL Anywhere databases, see "Initialization utility (dbinit)" [*SQL Anywhere Server - Database Administration*].

If you are using a SQL Anywhere database that was created before version 10.0.0, it must be upgraded.

For information about upgrading your database, see "Upgrading to SQL Anywhere 12" [*SQL Anywhere 12 - Changes and Upgrading*].

Note
The easiest way to create and maintain your server message store is in Sybase Central. From the QAnywhere 12 plug-in task pane, click **Server Message Store**.

Example

To create a SQL Anywhere database called *qanytest.db*, run the following command:

```
dbinit -s qanytest.db
```

Run the MobiLink setup script on the database:

```
%SQLANY12%\MobiLink\setup\syncsa.sql
```

This database is ready to use as a server message store.

Server Management Requests

A QAnywhere client application can send special messages to the server called **server management requests**. These messages contain content that is formatted as XML and are addressed to the QAnywhere

system queue. They require a special authentication string. Server management requests can perform a variety of functions, such as querying for active clients, querying message store properties, and querying messages.

For details about the functions you can perform with server management requests and how to use them, see "Server management request writing" on page 148.

Client message stores

The client message store can be a SQL Anywhere or UltraLite database on the remote device. SQL Anywhere databases are recommended for data synchronization applications. UltraLite databases are recommended for applications used exclusively for storing and forwarding messages. The application connects to this message store using the QAnywhere API. When using an UltraLite database as a client message store, the QAnywhere API accesses the store using the UltraLite Engine, and not the in-process UltraLite runtime.

The client message store must be used exclusively for QAnywhere applications. The QAnywhere message store database should not be accessed by any application other than QAnywhere applications using the QAnywhere API. However, you can run another database within the database server. This is useful if you have a QAnywhere client message store and a MobiLink synchronization client running on the same device.

Using a relational database as a message store provides a secure and high-performance store.

See "Creating a secure client message store" on page 115.

Setting up the client message store

Create a client message store

You can also upgrade a client message store that was created in a previous version of QAnywhere.

Note
The easiest way to create and maintain your client message store is in Sybase Central. From the QAnywhere 12 plug-in task pane, click **Client Message Store**.

1. Create a SQL Anywhere or UltraLite database.

   See "SQL Anywhere database creation" [*SQL Anywhere Server - Database Administration*].

2. Initialize each client message store by running the QAnywhere Agent or the QAnywhere UltraLite Agent with the following options:

   - **-c option** to specify a connection string to the database you just created.

     See "-c qaagent option" on page 675.

- **-si option** to initialize the database. The -si option creates a default database user and password. The agent shuts down after initializing the database.

  When you initialize QAnywhere by running qaagent with the -si option, the QAnywhere Agent creates client system tables that are required for QAnywhere messaging. QAnywhere also uses server system tables. These are created when you install MobiLink setup. All QAnywhere system table names begin ml_qa_ and cannot be altered.

  See "-si qaagent option" on page 690.

- **-id option** optionally, if you want to pre-assign a client message store ID.

  See "Creating client message store IDs" on page 24 and "-id qaagent option" on page 678.

- **-mu option** optionally, if you want to create a user name to use for authentication with the MobiLink server. If you do not use -mu at this point, you can specify it any time you start the QAnywhere Agent and the name is created if it does not already exist.

3. If you used the -mu option to create a user name, you need to add the name to the server message store. This can be done automatically using the mlsrv12 -zu+ option, or can be done in other ways.

   See "Registering QAnywhere client user names" on page 30.

4. Change the default passwords and take other steps to ensure that the client message store is secure.

   See "Creating a secure client message store" on page 115.

Creating client message store IDs

If you do not specify a client message store ID, then the first time you run qaagent after you run qaagent with -si, the device name is assigned as the client message store ID. The ID appears in the QAnywhere Agent window.

You may find it convenient to specify an ID manually. You can do so in the following ways:

- Specify the ID with the qaagent -id option when you use the qaagent -si option to initialize the client message store.
- Specify the ID with the -id option the first time you run qaagent after you initialize the client message store.

See "QAnywhere Agent utilities reference" on page 672.

Client message store IDs must differ by more than case. For example, don't have two message store IDs called AAA and aaa.

The client message store ID has a limit of 128 characters.

Transaction logs

It is recommended that you use a transaction log, both because a SQL Anywhere database runs most efficiently when using one and because transaction logs provide protection if there is database failure. However, the transaction log can grow very large. For this reason, the QAnywhere Agent by default sets

the dbsrv12 -m option, which causes the contents of the transaction log to be deleted at checkpoints. This is recommended. If you specify the StartLine parameter in the qaagent -c option, you should specify -m.

Protecting your client message stores

For information about backup and recovery, see "Backup and recovery plan design" [*SQL Anywhere Server - Database Administration*].

Example of creating a client message store

The following command creates a SQL Anywhere database called qanyclient.db. (The dbinit -i and -s options are not required, but are good practice on small devices.)

```
dbinit -i -s qanyclient.db
```

The following command connects to *qanyclient.db* and initializes it as a QAnywhere client database:

```
qaagent -si -c "DBF=qanyclient.db"
```

See "Initialization utility (dbinit)" [*SQL Anywhere Server - Database Administration*] and "QAnywhere Agent utilities reference" on page 672.

SQL Anywhere and UltraLite client differences

QAnywhere client applications can now use an UltraLite database as the client message store. This provides a lighter-weight solution for pure messaging applications on mobile devices. A pure messaging application means an application that uses store and forward messaging, but not data synchronization.

Some of the key advantages of UltraLite are:

- It has a smaller application footprint and does not require full SQL Anywhere installation.
- It has a smaller process footprint. QAnywhere Agent requires only 3 processes instead of 4 (uleng12, dblsn, and qauagent instead of dbeng12, dbmlsync, dblsn, and qaagent).

Keep in mind the following limitations of UltraLite when deciding between SQL Anywhere and UltraLite:

- There is no support for "ESCAPE" keyword in transmission rule condition syntax.
- There is limited support for property attributes. You can only use property attribute functionality with the predefined property ias_Network. See "Custom client message store properties" on page 718.

UltraLite should always be used rather than SQL Anywhere when SQL Anywhere is not already installed. SQL Anywhere is available for circumstances where you want to add messaging along side an already implemented SQL Anywhere data synchronization solution. However, UltraLite is recommended in all pure messaging environments.

From an application perspective, the client API remains the same for UltraLite as for SQL Anywhere with the following exception: the QAManager configuration properties should include the setting DATABASE_TYPE=UltraLite for UltraLite message stores. If the property DATABASE_TYPE is not set, the default is SQLAnywhere.

The client APIs supported for UltraLite are C# (for Microsoft .NET) and Java. For UltraLite, the C++ and SQL APIs are not supported.

The other difference from an application perspective is that the QAnywhere Agent for UltraLite is qauagent.exe. The QAnywhere Agent for UltraLite supports many of the same options as the QAnywhere Agent, with the following exceptions:

- -sv is not applicable to UltraLite
- -pc[+|-] is not supported by qauagent
- -sur is not applicable to UltraLite
- -c connection parameters are restricted to those documented in "UltraLite connection parameters" [*UltraLite - Database Management and Reference*]

In general, transmission rules are fully supported by the QAnywhere Agent for UltraLite. The one limitation is the support for Property attributes. Transmission rules can only use the following attributes of the predefined property ias_Network:

- ias_Network.Cost
- ias_Network.CommunicationAddress
- ias_Network.CommunicationType

See also

- "Custom client message store properties" on page 718
- "qauagent utility" on page 696

Client message store properties

There are two types of client message store property:

- **Predefined message store properties** These message store properties are always prefixed with ias_ or IAS_.
- **Custom message store properties** These are message store properties that you define. You cannot prefix them with ias_ or IAS_.

You can access client message store properties using the get and set methods defined in the appropriate class and pass the name of the predefined or custom property as the first parameter.

See "Client message store property management" on page 125.

You can also use message store properties in transmission rules, delete rules, and message selectors. See:

- "QAnywhere transmission and delete rules" on page 729

Predefined client message store properties

Several client message store properties have been predefined for your convenience. The predefined message store properties are:

- ias_Adapters
- ias_MaxDeliveryAttempts
- ias_MaxDownloadSize
- ias_MaxUploadSize
- ias_Network
- ias_Network.Adapter
- ias_Network.RAS
- ias_Network.IP
- ias_Network.MAC
- ias_RASNames
- ias_StoreID
- ias_StoreInitialized
- ias_StoreVersion

For details about client message store predefined properties, see:

- "Predefined client message store properties" on page 717

Custom client message store properties

QAnywhere allows you to define your own client message store properties using the QAnywhere C++, Java, SQL or .NET APIs. These properties are shared between applications connected to the same message store. They are also synchronized to the server message store so that they are available to server-side transmission rules for this client.

Client message store property names are case insensitive. You can use a sequence of letters, digits, and underscores, but the first character must be a letter. The following names are reserved and may not be used as message store property names:

- NULL
- TRUE
- FALSE
- NOT
- AND
- OR
- BETWEEN
- LIKE
- IN
- IS
- ESCAPE (SQL Anywhere message stores only)
- Any name beginning with **ias_**

Client message store properties can have attributes that you define. An attribute is defined by appending a dot after the property name followed by the attribute name. The main use of this feature is to be able to use information about your network in your transmission rules.

Limited support is provided for property attributes when using UltraLite as a client message store. UltraLite message stores only support the predefined ias_Network property.

For more details about using customer client message store properties, see:

- "Using custom client message store property attributes" on page 719
- "Predefined client message store properties" on page 717

QAnywhere messaging setup

The following sections describe how to set up QAnywhere messaging on the client and server, how to use pus notifications, and how to set up a failover mechanism.

Setting up server-side components

Overview of setting up QAnywhere server-side components

Note
The easiest way to create and maintain your server message store is in Sybase Central. From the QAnywhere 12 plug-in task pane, choose **Server Message Store**.

1. Set up a server message store and start it. This can be any MobiLink consolidated database.

   See "Server message store setup" on page 22.

2. Start mlsrv12 with the -m option and a connection to the server message store.

   See "Starting QAnywhere with MobiLink enabled" on page 29.

3. Add client user names to the server message store.

   See "Registering QAnywhere client user names" on page 30.

Starting QAnywhere with MobiLink enabled

QAnywhere uses MobiLink synchronization to transport messages. The QAnywhere server is a MobiLink server with messaging enabled.

To run the QAnywhere server, start the MobiLink server (mlsrv12) with the following options:

- **-c *connection-string*** Specifies the connection string to connect to the server message store. This is a required mlsrv12 option.

  See "-c mlsrv12 option" [*MobiLink - Server Administration*].

- **-m** Enables QAnywhere messaging.

You can also use other MobiLink server options to customize your operations.

Note
If you are integrating with a JMS messaging system, there are other options you must specify when you start the MobiLink server.

Example

To start QAnywhere messaging when you are using the sample server message store (*%SQLANYSAMP12%\QAnywhere\server\qanyserv.db*), run the following command:

```
mlsrv12 -m -c "dsn=QAnywhere 12 Demo"
```

The QAnywhere sample server message store uses the following ODBC data source: **QAnywhere 12 Demo**.

See also

- "Consolidated database setup" [*MobiLink - Server Administration*]
- "-m mlsrv12 option" [*MobiLink - Server Administration*]
- "mlsrv12 syntax" [*MobiLink - Server Administration*]
- "Starting the MobiLink server for JMS integration" on page 131

Registering QAnywhere client user names

Each QAnywhere client message store has a unique ID that identifies it. In addition, the client message store has a MobiLink user name that you can optionally use to authenticate your client message store with the MobiLink server. You can specify a MobiLink user name with the qaagent -mu option, or if you do not, one is created with the same name as your client message store ID.

You must register the MobiLink user name with the server message store. There are several methods for doing this:

- Use the mluser utility.
- Use the MobiLink 12 plug-in Sybase Central.
- Specify the -zu+ option with mlsrv12. In this case, any existing MobiLink users that have not been added to the consolidated database are added when they first synchronize. This is useful during development, but is not recommended for production environments.

For more information about MobiLink user names, see "MobiLink users" [*MobiLink - Client Administration*].

For more information about client message store IDs, see "-id qaagent option" on page 678.

See also

- "MobiLink User Authentication utility (mluser)" [*MobiLink - Server Administration*]
- "-zu mlsrv12 option" [*MobiLink - Server Administration*]

Setting properties for clients on the QAnywhere server

As a convenience, you can use the QAnywhere 12 plug-in to set properties for QAnywhere clients on the QAnywhere server. When you do this, you need to add the client to the server. The first time you synchronize to the client, the properties are downloaded.

Add a client user name using Sybase Central

1. Start Sybase Central:
   a. Click **Start** » **Programs** » **SQL Anywhere 12** » **Administration Tools** » **Sybase Central**.
   b. Click **Connections** » **Connect With QAnywhere 12**.
   c. Specify an **ODBC Data Source Name** or **ODBC Data Source File**, and the **User ID** and **Password** if required. Click **OK**.
2. Click **File** » **New** » **Client**.
3. Type the name of the client.
4. Click **OK**.

See also

- "Registering QAnywhere client user names" on page 30

Logging the QAnywhere server

The QAnywhere server is a MobiLink server with messaging enabled. The QAnywhere server log files are MobiLink log files.

MobiLink Server Log File Viewer

To view log files for the QAnywhere server, open Sybase Central and click **Tools** » **QAnywhere 12** » **MobiLink Server Log File Viewer**. You are prompted to choose a log file to view.

The Log Viewer reads information that is stored in MobiLink log files. It does not connect to the MobiLink server or change the composition of log files.

The Log Viewer allows you to filter the information that you view. In addition, it provides statistics based on the information in the log.

See also

- "Log MobiLink server actions" [*MobiLink - Server Administration*]

Relay Server use

To use a Relay Server for communication with the MobiLink server, configure the QAnywhere Agent to use HTTP or HTTPS as the network protocol.

For example:

```
qaagent -c
"dbf=mystore.db;server=mystore;dbn=mystore;uid=ml_qa_user;pwd=qanywhere"
     -x http(host=webserver01;port=80;url_suffix=/rs/client/rs_client.dll/
FarmName)
```

See also

- "Introduction to the Relay Server" [*Relay Server*]
- "Run the MobiLink server in a server farm" [*MobiLink - Server Administration*]

Setting up client-side components

Overview of setting up client-side components

Note
The easiest way to create and maintain your client message store is in Sybase Central. From the QAnywhere 12 plug-in task pane, click **Client Message Store**.

1. Create a SQL Anywhere database and initialize it as a client message store.

   See "Setting up the client message store" on page 23.

2. Write client applications.

   See "How to write QAnywhere client applications" on page 49.

3. Start the QAnywhere Agent.

   See "Starting the QAnywhere agent" on page 44.

Push notifications

A push notification is a special message delivered from the server message store to a QAnywhere client that prompts the client to initiate a message transmission. Push notification is on by default but is optional. Push notifications introduce extra components to the QAnywhere architecture:

- At the server, a QAnywhere Notifier sends push notifications.
- At the client, a QAnywhere Listener receives these push notifications and passes them on to the QAnywhere Agent.
- At the client, a notification of each push notification is sent to the system queue.

If you use the scheduled or automatic QAnywhere Agent policies, push notifications automatically cause clients to initiate message transmission. If you use the on demand policy, you must handle push requests manually using an event handler.

For more information about manually handling push notifications, see "Notifications of push notification" on page 60.

For more information about QAnywhere Agent policies, see "Determining when message transmission should occur on the client" on page 46.

Push notifications are enabled by default: the qaagent -push option is by default set to connected. In connected mode, push notifications are sent over TCP/IP persistent connection.

If you are using UDP, push notifications are likely to work without any configuration, but due to a limitation in the UDP implementation of ActiveSync, they do not work with ActiveSync.

See also

- "Scenario for messaging with push notifications" on page 6
- "Notifications of push notification" on page 60
- "-push qaagent option" on page 687

Push notification configuration

A push notification is a special message that is sent from the QAnywhere server to a QAnywhere client when a message arrives at the server message store that is destined for that client. The push notification is sent by a program called the **Notifier**, which runs on the server, and is received by a program called the **Listener**, which runs on the client. Push notifications are sent via a **gateway**. When the client receives the push notification, it initiates message transmission to receive messages waiting at the server or it takes some custom action.

Notifiers, Listeners and gateways are configured to work in QAnywhere without any modification. In rare circumstances, you may want to configure them. Also, there are some Notifier settings that you may want to change.

You can disable push notifications so as not to use Notifiers or Listeners. See "-push qaagent option" on page 687.

For information about the client's response to a push notification, see "Determining when message transmission should occur on the client" on page 46.

See also

- "Configuring the QAnywhere Notifier" on page 33
- "Configuring the MobiLink Listener" on page 35
- "Configuring QAnywhere gateways" on page 36

Configuring the QAnywhere Notifier

The QAnywhere Notifier is created by MobiLink setup scripts and is started when you run the MobiLink server with the -m option. The QAnywhere Notifier is called QAnyNotifier_client.

QAnyNotifier_client uses the defaults described in "MobiLink server settings for server-initiated synchronization" [*MobiLink - Server-Initiated Synchronization*], with the following exceptions:

- The gui property is set to off, meaning that the Notifier window is not displayed on the computer where the Notifier is running.
- The enable property is set to no, meaning that you have to run mlsrv12 with the -m option to start the Notifier.
- The poll_every property is set to 5, which means that the Notifier polls every five seconds to see if a push notification needs to be sent.

You can change the following Notifier properties:

- poll_every property
- resend interval in the request_cursor property
- time to live in the request_cursor property

Note
Other than the three properties listed here, you should not change any Notifier properties. Do not change any other columns in the request_cursor.

Poll_every property

You can change the default polling interval of QAnyNotifier_client by changing the value 5 in the following code and running it against your consolidated database:

```
CALL ml_add_property( 'SIS', 'Notifier(QAnyNotifier_client)', 'poll_every',
'5' )
```

See "Notifier properties" [*MobiLink - Server-Initiated Synchronization*].

Resend interval and time to live

The QAnywhere Notifier contains a default request_cursor. The request_cursor determines what information is sent in a push request, who receives the information, when, and where. You should not change any of the defaults except the resend interval and time to live. The resend interval specifies that an unreceived push notification should be re-sent every 5 minutes by default. The time to live specifies that an unreceived push notification is re-sent for 3 hours by default. These default settings are usually optimal. Following is the default request_cursor that is provided with QAnyNotifier_client:

```
SELECT
    u.user_id,
    'Default-DeviceTracker',
    'qa',
    u.name,
    u.name,
    '5M',
    '3H'
    FROM ml_qa_notifications u
    WHERE EXISTS(SELECT *
        FROM ml_listening l
        WHERE l.name = u.name AND l.listening = 'y')
```

For more information about the columns in the request_cursor, see "Push request requirements" [*MobiLink - Server-Initiated Synchronization*].

You can change the resend interval from the default of 5 minutes by changing the value 5M in the following code. You can change the time to live default of 3 hours by changing the value 3H.

```
CALL ml_add_property(
    'SIS',
    'Notifier(QAnyNotifier_client)',
    'request_cursor',
    'SELECT u.user_id,
        ''Default-DeviceTracker'',
        ''qa'',
        u.name,
        u.name,
        ''5M'',
        ''3H''
        FROM ml_qa_notifications u
        WHERE EXISTS(SELECT *
            FROM ml_listening l
            WHERE l.name = u.name AND l.listening = ''y'')'
)
```

For more information, see "request_cursor event" [*MobiLink - Server-Initiated Synchronization*].

See also

- "Notifier events and properties configuration" [*MobiLink - Server-Initiated Synchronization*]
- "MobiLink server settings for server-initiated synchronization" [*MobiLink - Server-Initiated Synchronization*]
- "Notifiers" [*MobiLink - Server-Initiated Synchronization*]
- "ml_add_property system procedure" [*MobiLink - Server Administration*]
- "Push requests" [*MobiLink - Server-Initiated Synchronization*]

Configuring the MobiLink Listener

The MobiLink Listener runs on the same device as the client message store. The MobiLink Listener receives push notifications from the server and passes them on to the QAnywhere Agent.

The MobiLink Listener is configured to work with QAnywhere. In some rare circumstances, you may want to change the default behavior.

For example, if you need to manually start the MobiLink Listener with custom options, you could do so with the following command:

```
dblsn.exe -u ias_mystore_lsn -e mystore -t+ mystore
          ... other options ...
```

For the QAnywhere Agent to find the MobiLink Listener you just started, you would also need to restart the QAnywhere Agent as follows:

```
qaagent -c "dbf=mystore.db;server=mystore;dbn=mystore" -id mystore
  -lp 5001 -x tcpip(host=acme.com)
```

where 5001 is the receiver port of the UDP listener library of dblsn.

See also

- "Listeners" [*MobiLink - Server-Initiated Synchronization*]
- "MobiLink Listener utility for Windows devices (dblsn)" [*MobiLink - Server-Initiated Synchronization*]
- "Configuring QAnywhere gateways" on page 36

Configuring QAnywhere gateways

Gateways are the way that push notifications are sent. By default, QAnywhere uses the default device tracker gateway. The device tracker gateway first tries to use the SYNC gateway, which uses the same protocol as is used for MobiLink synchronization and which is persistent. Usually the default device tracker gateway is the best way to send push notifications. However, you can also use an SMS or UDP gateway.

To configure a gateway, see "MobiLink server settings for server-initiated synchronization" [*MobiLink - Server-Initiated Synchronization*] and "Gateway properties" [*MobiLink - Server-Initiated Synchronization*].

To use an SMS gateway, you need to start the MobiLink Listener with new options. See "Configuring the MobiLink Listener" on page 35.

To use a UDP gateway, you need to set the -push disconnected option of qaagent. See "-push qaagent option" on page 687.

See also

- "Gateways and carriers" [*MobiLink - Server-Initiated Synchronization*]

Failover mechanism setup

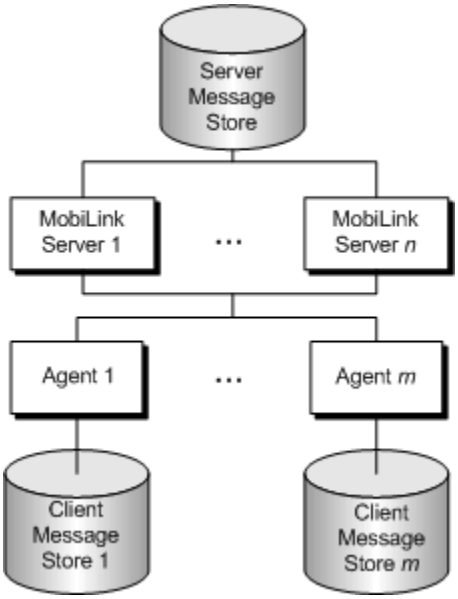
QAnywhere applications can be set up with a failover mechanism so that alternate MobiLink servers can be used if one fails. To support failover, each QAnywhere Agent must be started with a list of MobiLink servers. The first MobiLink server specified in the list is the primary server. The remaining servers in the list are alternate servers.

For example, running the following command on the remote device starts the QAnywhere Agent with one primary server and one alternate server:

```
qaagent -x tcpip(host=ml1.ianywhere.com)
        -x tcpip(host=ml2.ianywhere.com)
```

Each QAnywhere Agent can have a different primary server.

The following diagram describes a failover configuration in which you have multiple MobiLink servers and multiple QAnywhere agents. You have multiple client message stores, but all MobiLink servers are connected to the same server-side message store.



This configuration has the following characteristics:

- When a message transmission occurs, all messages in the server message store are delivered to the client message store regardless of the server that the QAnywhere Agent is connected to.
- Push Notifications are sent to a QAnywhere Agent only when the QAnywhere Agent is connected to its primary server.
- There is a single point of failure. If the computer with the server message store is unavailable, no messaging can take place.

By default, when you set up failover MobiLink servers, the QAnywhere Agent always tries an alternate server immediately upon a failure to reach the primary server. If you want to change this default behavior, you can use the QAnywhere Agent -fr option to cause the QAnywhere Agent to try the primary server again before going to the alternate server, and to specify the number of times it should retry. You can use the -fd option to specify the amount of time between retries of the primary server.

The -fr and -fd options apply only to the primary server. If a connection to the primary server cannot be established after the specified number of attempts, the QAnywhere Agent tries to connect to an alternate server. The Agent attempts to connect to each alternate server only once. An error is issued if the Agent cannot establish a connection to an alternate server.

See also

- "-x qaagent option" on page 694
- "-fd qaagent option" on page 677
- "-fr qaagent option" on page 677
- "Starting the QAnywhere agent" on page 44

The QAnywhere agent

The QAnywhere Agent (qaagent) is a separate process running on the client device. It monitors the client message store and determines when message transmission should occur.

The QAnywhere Agent transmits messages between the server message store and the client message store. You can run multiple instances of the QAnywhere Agent on the same device, but each instance must be connected to its own message store. Each message store must have a unique message store ID.

Message transmission policies

QAnywhere clients use policies to determine when to perform a message transmission. Policies include:

- **"Scheduled policy" on page 39** The scheduled policy initiates message transmission at a specified time interval.
- **"Automatic policy" on page 40** The automatic policy initiates a message transmission whenever a message to or from the client is ready for delivery.
- **"On demand policy" on page 41** The on demand policy permits the user to control message transmission.
- **"Custom policy" on page 41** The custom policy uses transmission rules to add further control over message transmission.

Scheduled policy

The scheduled policy instructs the Agent to perform a transmission at a specified time interval.

To invoke a schedule, click **scheduled** in the **Command File Properties** window or specify the keyword when you start the QAnywhere Agent:

qaagent -policy scheduled [*interval*] ...

where *interval* is in seconds.

The default is 900 seconds (15 minutes).

When a schedule is specified, every *n* seconds the Agent performs message transmission if any of the following conditions are met:

- New messages were placed in the client message store since the previous time interval elapsed.
- A message status change occurred since the previous time interval elapsed. This typically occurs when a message is acknowledged by the application.

For more information about acknowledgement, see:

- "AcknowledgementMode enumeration [QAnywhere .NET]" on page 302
- "AcknowledgementMode class [QAnywhere C++]" on page 354
- "AcknowledgementMode interface [QAnywhere Java]" on page 468

- A push notification was received since the previous time interval elapsed.
- A network status change notification was received since the previous time interval elapsed.
- Push notifications are disabled.

You can call the trigger send/receive method to override the time interval. It forces message transmission to occur before the time interval elapses.

See:

- "QAManagerBase.TriggerSendReceive method [QAnywhere .NET]" on page 265
- "QAManagerBase.triggerSendReceive method [QAnywhere C++]" on page 424
- "QAManagerBase.triggerSendReceive method [QAnywhere Java]" on page 545
- SQL: "ml_qa_triggersendreceive" on page 655

Automatic policy

The automatic policy attempts to keep the client and server message stores as up-to-date as possible by synchronizing whenever a message is sent or received. This policy is not recommended for applications that frequently send and receive messages.

When using the automatic policy, message transmission is performed when any of the following conditions occurs:

- PutMessage() is called.

  See:

  - "QAManagerBase.PutMessage method [QAnywhere .NET]" on page 251
  - "QAManagerBase.putMessage method [QAnywhere C++]" on page 416
  - "QAManagerBase.putMessage method [QAnywhere Java]" on page 534
  - SQL: "ml_qa_putmessage" on page 654

- A message status changes has occurred. This typically occurs when a received message is acknowledged by the application.

  See:

  - "AcknowledgementMode enumeration [QAnywhere .NET]" on page 302
  - "AcknowledgementMode class [QAnywhere C++]" on page 354
  - "AcknowledgementMode interface [QAnywhere Java]" on page 468
  - SQL: all messaging using the SQL API is transactional

- A Push Notification is received.

See “Push notifications” on page 32.

- A Network Status Change Notification is received.

  See “Notifications of push notification” on page 60.

- TriggerSendReceive() is called.

  See:

  - “QAManagerBase.TriggerSendReceive method [QAnywhere .NET]” on page 265
  - “QAManagerBase.triggerSendReceive method [QAnywhere C++]” on page 424
  - “QAManagerBase.triggerSendReceive method [QAnywhere Java]” on page 545
  - SQL: “ml_qa_triggersendreceive” on page 655

On demand policy

The on demand policy causes message transmission to occur only when instructed to do so by an application.

An application forces a message transmission to occur by calling TriggerSendReceive().

When the agent receives a Push Notification or a Network Status Change Notification, a corresponding message is sent to the system queue. This allows an application to detect these events and force a message transmission by calling TriggerSendReceive().

See:

- “QAManagerBase.TriggerSendReceive method [QAnywhere .NET]” on page 265
- “QAManagerBase.triggerSendReceive method [QAnywhere C++]” on page 424
- “QAManagerBase.triggerSendReceive method [QAnywhere Java]” on page 545
- SQL: “ml_qa_triggersendreceive” on page 655

For more information about handling push notifications and network status changes, see “System queue” on page 58.

Custom policy

A custom policy allows you to define when message transmission occurs and which messages to send in the message transmission.

When creating custom policy rules for your application, it is recommended that you include a default all-inclusive rule so that messages are not accidentally overlooked by other rules. For example, this rule synchronizes messages that are at least one day old,

```
auto=DATEADD( day, 1, ias_StatusTime ) < ias_CurrentTimestamp
```

The following is a list of factors that impact the effectiveness of synchronization and should be considered when creating your own custom policy rules.

- Message sizes
- Synchronization frequency
- Bandwidth and network reliability
- Priority messaging
- Data transfer costs

The custom policy is defined by a set of transmission rules.

Each rule is of the following form:

schedule = *condition*

where *schedule* defines when *condition* is evaluated. For more information, see "Rule syntax" on page 729.

All messages satisfying *condition* are transmitted. In particular, if *schedule* is automatic, the condition is evaluated when any of the following conditions occurs:

- PutMessage() is called.

  See:

  - "QAManagerBase.PutMessage method [QAnywhere .NET]" on page 251
  - "QAManagerBase.putMessage method [QAnywhere C++]" on page 416
  - "QAManagerBase.putMessage method [QAnywhere Java]" on page 534
  - SQL: "ml_qa_putmessage" on page 654

- A message status change has occurred. This typically occurs when a message is acknowledged by the application.

  See:

  - "AcknowledgementMode enumeration [QAnywhere .NET]" on page 302
  - "AcknowledgementMode class [QAnywhere C++]" on page 354
  - "AcknowledgementMode interface [QAnywhere Java]" on page 468
  - SQL: all messaging using the SQL API is transactional

- A Push Notification is received.

  See "Push notifications" on page 32.

- A Network Status Change Notification is received.
- TriggerSendReceive ( ) is called.

See:

- "QAManagerBase.TriggerSendReceive method [QAnywhere .NET]" on page 265
- "QAManagerBase.triggerSendReceive method [QAnywhere C++]" on page 424
- "QAManagerBase.triggerSendReceive method [QAnywhere Java]" on page 545
- SQL: "ml_qa_triggersendreceive" on page 655

Transmission status

The easiest way to determine the transmission status of a message is using Sybase Central. Go to the **General** tab of the **Message Properties** window to view the message transmission status. The possible values are:

- **transmitted** The message has been sent.
- **transmitting** The message is in the process of being sent.
- **untransmitted** The message has not been sent.
- **do_not_transmit** The message should not be sent.

Message status

The easiest way to determine the status of a message is using Sybase Central. Go to the **General** tab of the **Message Properties** window to view the message status. The possible values are:

- **Pending** The message has been sent but not received.
- **Receiving** The message is in the process of being received, or it was received but not acknowledged.
- **Final** The message has achieved a final state.
- **Expired** The message was not received before its expiration time has passed.
- **Cancelled** The message has been canceled.
- **Unreceivable** The message is either malformed, or there were too many failed attempts to deliver it.
- **Received** The message has been received and acknowledged.

Transmission rules

Message transmission is the action of moving messages from a client message store to a server message store, or vice versa.

Message transmission is handled by the QAnywhere Agent and the MobiLink server:

- The QAnywhere Agent is connected to the client message store. It transmits messages to and from the MobiLink server.
- The MobiLink server is connected to the server message store. It receives message transmissions from QAnywhere Agents and transmits them to other QAnywhere Agents.

Message transmission can only take place between a client message store and a server message store. A message transmission can only occur when a QAnywhere Agent is connected to a MobiLink server.

In QAnywhere, rules are logic that determines when message transmission is to occur, which messages to transmit, and when messages should be deleted. You can specify rules on the client and on the server.

Rules have two parts: a schedule and a condition. The schedule defines when an event is to occur. The condition defines which messages are to be part of the event. For example, if the event is message transmission, then the schedule indicates when transmission occurs and the condition defines which messages are included in the transmission. If the event is message deletion, then the schedule indicates when deleting occurs and the condition indicates which messages are deleted.

Transmission rules allow you to specify when message transmission is to occur and which messages to transmit. You can specify transmission rules for both the client and the server.

See also

- “Client transmission rules” on page 738
- “Server transmission rules” on page 739
- “Rule syntax” on page 729
- “Rule variables” on page 735

Delete rules

Delete rules allow you to specify when messages should be deleted from the message stores, if you do not want to use the default behavior. You can specify delete rules for both the client and the server.

See also

- “Message delete rules” on page 741

Starting the QAnywhere agent

You can run the Agent on the command line using command line options. At a minimum, you need to start the Agent with the following options:

- **Connection parameters** to connect to the client message store.

  In the **Agent Configuration File Properties** window, this is the information on the **Message Store** tab.

In the qaagent command line, this is specified with the -c option.

See "-c qaagent option" on page 675.

- **Client message store ID** to identify the client message store. The first time you run qaagent after you have initialized a client message store, you can optionally use this option to name the message store; if you do not, the device name is used by default. After that, you must use the -id option every time you start qaagent to specify a unique client message store ID.

  In the **Agent Configuration File Properties** window, this is specified on the **General** tab.

  In the qaagent command line, this is specified with the -id option.

  See "-id qaagent option" on page 678.

- **Network protocol and protocol options** to connect to the MobiLink server. This is required unless the MobiLink server is running on the same device as the QAnywhere agent and default communication parameters are used.

  In the **Agent Configuration File Properties** window, this is the server information on the **Server** tab.

  In the qaagent command line, this is the -x option.

  See "-x qaagent option" on page 694.

For a complete list of all QAnywhere Agent options, see "qaagent utility" on page 672.

Starting qaagent on Windows Mobile

On Windows Mobile, you might want to start the QAnywhere Agent in quiet mode by specifying the -qi option.

See "-qi qaagent option" on page 689.

Running multiple instances of QAnywhere Agent

You can run multiple instances of qaagent on a device. However, when you start a second instance:

- The second instance of QAnywhere Agent must be started with a different database file.
- You must specify a unique message store ID using the -id option.

  See "-id qaagent option" on page 678.

Stopping the QAnywhere Agent

To stop the QAnywhere Agent, click **Shut Down** on the QAnywhere Agent messages window.

When you start the QAnywhere Agent in quiet mode, you can only stop it by running **qastop**.

See also

- “qastop utility” on page 717
- “-qi qaagent option” on page 689

Processes started by the QAnywhere Agent

The QAnywhere Agent starts other processes to handle various messaging tasks. Each of these processes is managed by the QAnywhere Agent, and does not need to be managed separately. When you start the QAnywhere Agent, it spawns the following processes:

- **dbmlsync** The dbmlsync executable is the MobiLink synchronization client. The dbmlsync executable is used to send and receive messages.

  Caution
  Do not run dbmlsync on a QAnywhere message store independently of qaagent.

- **dblsn** The dblsn executable is the MobiLink Listener utility. It receives push notifications. If you are not using push notifications, you do not need to supply the dblsn executable when you deploy your application, and you must run qaagent with -push **none**.

- **database server** The client message store is a SQL Anywhere or UltraLite database. QAnywhere Agent requires the database server to run the database. For Windows Mobile, the database server is *dbsrv12.exe*. For Windows, the database server is the personal database server *dbeng12.exe*.

  The QAnywhere Agent can spawn a database server or connect to a running server, depending on the communication parameters that you specify in the qaagent -c option.

See also

- “-push qaagent option” on page 687
- “-c qaagent option” on page 675

QAnywhere Agent deployment

For deployment information, see “QAnywhere application deployment”.

Determining when message transmission should occur on the client

On the client side, you determine when message transmission should occur by specifying **policies**. A policy tells the QAnywhere Agent when a message should be moved from the client message store to the server message store. If you do not specify a policy, transmission occurs automatically when a message is queued for delivery to the server by default. There is a custom policy and three predefined policies: scheduled, automatic, and on demand.

You can specify policies in the following ways:

- Using the QAnywhere 12 plug-in for Sybase Central, click **Tools** » **QAnywhere 12** » **New Agent Configuration File for SQL Anywhere**. Policies are specified on the **General** tab of the **Agent Configuration File Properties** window. This task creates a file with a *.qaa* extension, a Sybase Central convention.

  To specify custom properties using the QAnywhere 12 plug-in for Sybase Central, click **Tools** » **QAnywhere 12** » **New Agent Rule File**. This task creates a file with a *.qar* extension, a Sybase Central convention.

- Run qaagent on the command line using the -policy option. For custom policies, create a rules file and specify it.

See also

- "Message transmission policies" on page 39
- "Transmission rules" on page 43
- "-policy qaagent option" on page 685

Unreliable networks

Without incremental upload and download, messages are sent as a single piece, so if network connectivity is lost while a message is being uploaded or downloaded, transmission of the message fails. With incremental upload and download, large messages are broken into smaller message pieces. By allowing messages to be sent in smaller pieces, each message piece can be sent separately, resulting in the gradual upload or download of the message over several synchronizations. The complete message arrives at its destination once all of its message pieces have arrived.

For information about how to implement incremental uploads, see "-iu qaagent option" on page 680.

For information about how to implement incremental downloads, see "-idl qaagent option" on page 679.

How to write QAnywhere client applications

The following sections discuss how to write QAnywhere client applications, including an introduction to the various interfaces available for use with QAnywhere as well as an overview of the steps involved.

The QAnywhere interfaces

QAnywhere client applications manage the receiving and sending of QAnywhere messages. The applications can be written using one of several QAnywhere APIs:

- QAnywhere .NET API
- QAnywhere C++ API
- QAnywhere Java API
- QAnywhere SQL API

You can use a combination of client types in your QAnywhere system. For example, messages that are generated using QAnywhere SQL can also be received by a client created using the APIs for .NET, C++, or Java. If you have configured a JMS connector on your server, the messages can also be received by JMS clients. Similarly, QAnywhere SQL can be used to receive messages that were generated by QAnywhere .NET, C++, Java, or JMS clients.

QAnywhere .NET API

The QAnywhere .NET API is a programming interface for deployment to Windows computers using the Microsoft .NET Framework and to handheld devices running the Microsoft .NET Compact Framework. The QAnywhere .NET API is provided as the iAnywhere.QAnywhere.Client namespace.

QAnywhere supports Microsoft Visual Studio.

Note
In this document, code samples for the .NET API use the C# programming language, but the API can be accessed using any programming language that Microsoft .NET supports.

Versions of the TestMessage sample application are written in Java, C#, and Visual Basic .NET. There is also a .NET compact framework sample.

For more information about the .NET version of the TestMessage sample application, see “Lesson 4: Exploring the TestMessage client source code” on page 176.

See “QAnywhere .NET API reference for clients” on page 181.

QAnywhere C++ API

The QAnywhere C++ API supports Microsoft Visual Studio.

The QAnywhere C++ API consists of the following files:

- A set of header files (the main one being *qa.hpp*), located in *%SQLANY12%\sdk\include*.
- An import library (*qany12.lib*), located in *%SQLANY12%\sdk\lib\x86*, and *%SQLANY12%\sdk\lib\ce\arm.50*.
- A run-time DLL (*qany12.dll*) located in *%SQLANY12%\bin32*, and *%SQLANY12%\ce\arm.50*.

To access the API, your source code file must include the header file. The import library is used to link your application to the run-time DLL. The run-time DLL must be deployed with your application.

A version of the TestMessage sample application written in C++ is supplied in *%SQLANYSAMP12%\QAnywhere\Windows\MFC*.

See "QAnywhere C++ API reference for clients" on page 354.

QAnywhere Java API

The QAnywhere Java API supports JRE 1.4.2 and up. The Mobile web services wsdl compiler generates Java classes compatible with JDK 1.5.0 and up.

The QAnywhere Java API consists of the following files:

- API reference material, available in this book or in Javadoc format in the *documentation\en\javadocs\QAnywhere* subdirectory of your SQL Anywhere 12 installation.
- Runtime DLL (*qadbiuljni12.dll*) for UltraLite message stores, located in the *bin32* subdirectory of your SQL Anywhere 12 installation.
- An archive of the class files (*qaclient.jar*), located in the *java* subdirectory of your SQL Anywhere 12 installation.

The class file archive must be included in your path when you compile your application. The runtime DLL must be deployed with your application.

A version of the TestMessage sample application written in Java is supplied in *%SQLANYSAMP12%\QAnywhere\Java*.

See "QAnywhere Java API reference for clients" on page 468.

QAnywhere SQL API

The QAnywhere SQL API is a set of stored procedures that implement a messaging API in SQL. Using the QAnywhere SQL API, you can create messages, set or get message properties and content, send and receive messages, trigger message synchronization, and set and get message store properties.

See "QAnywhere SQL API reference" on page 618.

JMS connector

QAnywhere includes a JMS connector that provides connectivity between QAnywhere and JMS applications.

See:

- “Scenario for messaging with external messaging systems” on page 7
- “JMS connectors” on page 129
- “Tutorial: Using JMS connectors” on page 140

Mobile web services connector

QAnywhere includes a mobile web services connector for messaging between QAnywhere and web services.

See “Mobile web services” on page 89.

Quick start to writing a client application

Overview of setting up a client application

1. Initialize the appropriate QAnywhere API. See:
   - “Setting up .NET applications” on page 52
   - “Setting up C++ applications” on page 54
   - “Setting up Java applications” on page 55
   - “SQL application setup” on page 56
2. Set QAnywhere manager configuration properties. See “QAnywhere manager configuration properties” on page 80.
3. Write application code and compile. See:
   - “QAnywhere messages” on page 13
   - “Client message store properties” on page 26
   - “Sending QAnywhere messages” on page 61
   - “Receive QAnywhere messages” on page 68
   - “Very large messages” on page 72
   - “Transactional messaging implementation” on page 63
   - “Shutting down QAnywhere” on page 79
4. Deploy the application to the target device.

   See “QAnywhere application deployment” on page 111.

Other resources for getting started

- “Tutorial: Exploring TestMessage” on page 171
- Sample applications are installed to *%SQLANYSAMP12%\QAnywhere*.

Initializing a QAnywhere API

Before you can send or receive messages using QAnywhere, you must complete the following initialization tasks.

Setting up .NET applications

Before you can send or receive messages using QAnywhere .NET clients, you must complete the following initialization tasks.

You must make two changes to your Visual Studio project to be able to use it:

- Add a reference to the QAnywhere .NET DLL. Adding a reference tells Visual Studio .NET which DLL to include to find the code for the QAnywhere .NET API.
- Add a line to your source code to reference the QAnywhere .NET API classes. To use the QAnywhere .NET API, you must add a line to your source code to reference the data provider. You must add a different line for C# than for Visual Basic .NET.

In addition, you must initialize the QAnywhere .NET API.

Add a reference to the QAnywhere .NET API in a Visual Studio project

1. Start Visual Studio and open your project.
2. In the **Solution Explorer** window, right-click the **References** folder and click **Add Reference**.
3. On the **.NET** tab, click **Browse** to locate *iAnywhere.QAnywhere.Client.dll*. The default locations are:
   - .NET Framework 2.0: *%SQLANY12%\Assembly\V2*
   - .NET Compact Framework 2.0: *%SQLANY12%\ce\Assembly\V2*

   Select the DLL and click **Open**.
4. You can verify that the DLL is added to your project. Open the **Add Reference** window and then click the **.NET** tab. *iAnywhere.QAnywhere.Client.dll* appears in the **Selected Components** list. Click **OK** to close the window.

Referencing the data provider classes in your source code

Reference the QAnywhere .NET API classes in your code

1. Start Visual Studio and open your project.
2. If you are using C#, add the following line to the list of using directives at the beginning of your file:

   ```
   using iAnywhere.QAnywhere.Client;
   ```
3. If you are using Visual Basic, add the following line to the list of imports at the beginning of your file:


```
Imports iAnywhere.QAnywhere.Client
```

This line is not strictly required. However, it allows you to use short forms for the QAnywhere classes. Without it, you can still use the fully qualified class name in your code. For example:

```
iAnywhere.QAnywhere.Client.QAManager
mgr =
iAnywhere.QAnywhere.Client.QAManagerFactory.Instance.CreateQAManager(
"qa_manager.props" );
```

instead of

```
QAManager mgr = QAManagerFactory.Instance.CreateQAManager(
 "qa_manager.props" );
```

Initialize the QAnywhere .NET API

1. Include the iAnywhere.QAnywhere.Client namespace, as described in the previous procedure.

   ```
   using iAnywhere.QAnywhere.Client;
   ```

2. Create a QAManager object.

   For example, to create a default QAManager object, invoke CreateQAManager:

   ```
   QAManager mgr;
   mgr = QAManagerFactory.Instance.CreateQAManager();
   ```

   Tip
   For maximum concurrency benefits, multi-threaded applications should create a QAManager for each thread. See “Multi-threading considerations” on page 80.

   You can alternatively create a QAManager object that is customized using a properties file. The properties file is specified in the CreateQAManager method:

   ```
   mgr = QAManagerFactory.Instance.CreateQAManager(
     "qa_mgr.props" );
   ```

   where *qa_mgr.props* is the name of the properties file that resides on the remote device.

3. Initialize the QAManager object. For example:

   ```
   mgr.Open(
      AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT);
   ```

   The argument to the open method is an acknowledgement mode, which indicates how messages are to be acknowledged. It must be one of IMPLICIT_ACKNOWLEDGEMENT or EXPLICIT_ACKNOWLEDGEMENT. With implicit acknowledgement, messages are acknowledged when they are received by the client. With explicit acknowledgement, you must call the Acknowledge method on the QAManager to acknowledge the message.

   You are now ready to send messages.

Note
Instead of creating a QAManager, you can create a QATransactionalManager.

See also

- "QAnywhere .NET API reference for clients" on page 181
- "QAManagerFactory class [QAnywhere .NET]" on page 266
- "Implementing transactional messaging for .NET clients" on page 63
- "AcknowledgementMode enumeration [QAnywhere .NET]" on page 302

Setting up C++ applications

Before you can send or receive messages using QAnywhere C++ clients, you must complete the following initialization tasks.

Initialize the QAnywhere C++ API

Note
Instead of creating a QAManager, you can create a QATransactionalManager. See "Implementing transactional messaging for C++ clients" on page 64.

1. Include the QAnywhere header file.

   ```
   #include <qa.hpp>
   ```

   qa.hpp defines the QAnywhere classes.

2. Initialize QAnywhere.

   To do this, initialize a factory for creating QAManager objects.

   ```
   QAManagerFactory * factory;

   factory = QAnywhereFactory_init();
   if( factory == NULL ) {
         // Fatal error.
   }
   ```

   For more information about QAManagerFactory, see "QAManagerFactory class [QAnywhere C++]" on page 425.

3. Create a QAManager instance.

   You can create a default QAManager object as follows:

   ```
   QAManager *  mgr;

   // Create a manager
   mgr = factory->createQAManager();
   if( mgr == NULL ) {
     // fatal error
   }
   ```


See "QAManager class [QAnywhere C++]" on page 388.

Tip
For maximum concurrency benefits, multi-threaded applications should create a QAManager for each thread. See "Multi-threading considerations" on page 80.

You can customize a QAManager object programmatically or using a properties file.

- To customize QAManager programmatically, use setProperty().

  See "Setting QAnywhere manager configuration properties programmatically" on page 83.
- To use a properties file, specify the properties file in createQAManager():

  ```
  mgr = factory->createQAManager( "qa_mgr.props" );
  ```

  where *qa_mgr.props* is the name of the properties file on the remote device.

  See "QAnywhere manager configuration property settings in a file" on page 82.

4. Initialize the QAManager object.

   ```
   qa_bool  rc;
   rc=mgr->open(
        AcknowledgementMode::IMPLICIT_ACKNOWLEDGEMENT );
   ```

   The argument to the open method is an acknowledgement mode, which indicates how messages are to be acknowledged. It must be one of **IMPLICIT_ACKNOWLEDGEMENT** or **EXPLICIT_ACKNOWLEDGEMENT**. With implicit acknowledgement, messages are acknowledged when they are received by the client. With explicit acknowledgement, you must call one of the acknowledge methods on the QAManager to acknowledge the message.

   For more information about acknowledgement modes, see "AcknowledgementMode class [QAnywhere C++]" on page 354.

You are now ready to send messages.

See also

- "QAnywhere C++ API reference for clients" on page 354

Setting up Java applications

Before you can send or receive messages using QAnywhere Java clients, you must complete the following initialization tasks.

Initialize the QAnywhere Java API

Note
Instead of creating a QAManager, you can create a QATransactionalManager.

1. Add the location of *qaclient.jar* to your classpath. By default, the file is located in *%SQLANY12%\Java*.

2. Import the ianywhere.qanywhere.client package.

   ```
   import ianywhere.qanywhere.client.*;
   ```

3. Create a QAManager object.

   ```
   QAManager mgr;
   mgr = QAManagerFactory.getInstance().createQAManager(null);
   ```

   You can also customize a QAManager object by specifying a properties file to the createQAManager method:

   ```
   mgr = QAManagerFactory.getInstance().createQAManager("qa_mgr.props.");
   ```

   Tip
   For maximum concurrency benefits, multi-threaded applications should create a QAManager for each thread. See "Multi-threading considerations" on page 80.

4. Initialize the QAManager object.

   ```
   mgr.open(AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT);
   ```

   The argument to the open method is an acknowledgement mode, which indicates how messages are to be acknowledged. It must be one of IMPLICIT_ACKNOWLEDGEMENT or EXPLICIT_ACKNOWLEDGEMENT. With implicit acknowledgement, messages are acknowledged when they are received by the client. With explicit acknowledgement, you must call one of the acknowledge methods on the QAManager to acknowledge the message.

   You are now ready to send messages.

See also

- "QAnywhere Java API reference for clients" on page 468
- "Implementing transactional messaging for Java clients" on page 66
- "AcknowledgementMode interface [QAnywhere Java]" on page 468

SQL application setup

QAnywhere SQL allows you to perform, in SQL, much of the messaging functionality of the QAnywhere .NET, C++, and Java APIs. This functionality includes creating messages, setting or getting message properties and content, sending and receiving messages, triggering message synchronization, and setting and getting message store properties.

Messages that are generated with QAnywhere SQL can also be received by clients created with the programming APIs. If you have configured a JMS connector on your server, the messages can also be received by JMS clients. Similarly, QAnywhere SQL can be used to receive messages that were generated by QAnywhere .NET, C++, or Java API, or JMS clients.

QAnywhere SQL messaging coexists with user transactions. This means that committing a transaction commits all the QAnywhere operations on that connection.

Permissions

Only users with DBA privilege have automatic permission to execute the QAnywhere stored procedures. To give permission to a user, a user with DBA privilege must call the procedure ml_qa_grant_messaging_permissions.

Acknowledgement modes

The QAnywhere SQL API does not support IMPLICIT_ACKNOWLEDGEMENT or EXPLICIT_ACKNOWLEDGEMENT modes. All messaging through the SQL API is transactional.

Example

The following example creates a trigger on an inventory table. The trigger sends a message when the inventory for an item falls below a certain threshold. The message is sent after the transaction invoking the trigger is committed. If the transaction is rolled back, the message is not sent.

```
CREATE TRIGGER inventory_trigger AFTER UPDATE ON inventory
REFERENCING old AS oldinv new AS newinv
FOR EACH ROW
begin
   DECLARE msgid VARCHAR(128);
   IF oldinv.quantity > newinv.quantity AND newinv.quantity < 10 THEN
      -- Create the message
      SET msgid = ml_qa_createmessage();
      -- Set the message content
      CALL ml_qa_settextcontent( msgid,
         'Inventory of item ' || newinv.itemname
            || ' has fallen to only ' || newinv.quantity );
      -- Make the message high priority
      CALL ml_qa_setpriority( msgid, 9 );
      -- Set a message subject
      CALL ml_qa_setstringproperty( msgid,
         'tm_Subject', 'Inventory low!' );
      -- Send the message to the inventoryManager queue
      CALL ml_qa_putmessage( msgid,
         'inventoryManager' );
   end if;
end
```

See also

- “QAnywhere SQL API reference” on page 618
- “How to write QAnywhere client applications” on page 49
- “ml_qa_grant_messaging_permissions” on page 653

QAnywhere message addresses

A QAnywhere message address has two parts, the client message store ID and the application queue name:

```
id\queue-name
```

The queue name is specified inside the application, and must be known to instances of the sending application on other devices.

Each address can have at most one application associated with it at a time. More than one application running with the same address can result in undefined behavior during message retrieval.

When constructing addresses as strings in an application, be sure to escape the backslash character if necessary. Follow the string escaping rules for the programming language you are using. If your JMS destination contains a backslash, you must escape it with another backslash.

The address cannot be longer than 255 characters.

System queue

Notifications and network status changes are both sent to QAnywhere applications as **system messages**. System messages are the same as other messages, but are received in a separate queue named **system**.

See "System queue" on page 58.

Sending a message to a JMS connector

A QAnywhere-to-JMS destination address has two parts:

- The connector address. This is the value of the ianywhere.connector.address property.
- The JMS queue name. This is a queue that you create using your JMS administration tools.

The form of the destination address is:

```
connector-address\JMS-queue-name
```

For more information about addressing messages in a JMS application, see:

- "Setting up the client message store" on page 23
- "Sending a QAnywhere message to a JMS connector" on page 133
- "JMS connector property configuration" on page 132
- "Addressing JMS messages meant for QAnywhere" on page 135
- "Connectors" on page 129

System queue

A special queue called **system** exists to receive QAnywhere system messages. There are two types of message that are sent to the system queue:

- "Network status notifications"
- "Notifications of push notification"

Example

The following C# code processes system and normal messages and can be useful if you are using an on demand policy. It assumes that you have defined the message handling methods onMessage() and onSystemMessage() that implement the application logic for processing the messages.

```
// Declare the message listener and system listener.
private QAManager.MessageListener _receiveListener;
private QAManager.MessageListener _systemListener;
...

// Create a MessageListener that uses the appropriate message handlers.
_receiveListener = new QAManager.MessageListener( onMessage );
_systemListener = new QAManager.MessageListener( onSystemMessage );
...

// Register the message handler.
mgr.SetMessageListener( queue-name, _receiveListener );
mgr.SetMessageListener( "system", _systemListener );
```

The system message handler may query the message properties to identify what information it contains. The message type property indicates if the message holds a network status notification. For example, for a message `msg`, you could perform the following processing:

```
msg_type = (MessageType)msg.GetIntProperty( MessageProperties.MSG_TYPE );
if( msg_type == MessageType.NETWORK_STATUS_NOTIFICATION ) {
  // Process a network status change.
  mgr.TriggerSendReceive( );
} else if ( msg_type == MessageType.PUSH_NOTIFICATION ) {
  // Process a push notification.
  mgr.TriggerSendReceive( );
} else if ( msg_type == MessageType.REGULAR ) {
  // This message type should not be received on the
  // system queue. Take appropriate action here.
}
```

Network status notifications

When there is a change in network status, a message of type NETWORK_STATUS_NOTIFICATION is sent to the system queue. It has an expiry of one minute. This expiry time cannot be changed.

When a device goes into network coverage or out of network coverage, a message is sent to the system queue that contains the following information:

- **ias_Adapters** String. A list of network adapters that can be used to connect to the MobiLink server. The list is delimited by a vertical bar. This property can be read but should not be set. See:
  - “MessageProperties.ADAPTER field [QAnywhere .NET]” on page 183
  - “MessageProperties.ADAPTER variable [QAnywhere C++]” on page 358
  - “MessageProperties.ADAPTER variable [QAnywhere Java]” on page 471

- **ias_RASNames** String. A list of network names that can be used to connect to the MobiLink server. The list is delimited by a vertical bar. See:
  - "MessageProperties.RASNAMES field [QAnywhere .NET]" on page 188
  - "MessageProperties.RASNAMES variable [QAnywhere C++]" on page 362
  - "MessageProperties.RASNAMES variable [QAnywhere Java]" on page 475
- **ias_NetworkStatus** Int. The state of the network connection. The value is 1 if connected, 0 otherwise. See:
  - "MessageProperties.NETWORK_STATUS field [QAnywhere .NET]" on page 186
  - "MessageProperties.NETWORK_STATUS variable [QAnywhere C++]" on page 361
  - "MessageProperties.NETWORK_STATUS variable [QAnywhere Java]" on page 474

Monitoring network availability

You can use network status notifications to monitor network availability and take action when a device comes into coverage. For example, use the on demand policy and call QAManagerBase triggerSendReceive when a system queue message is received of type NETWORK_STATUS_NOTIFICATION with ias_NetworkStatus=1.

See also

- ias_MessageType in "Predefined message properties" on page 659
- "System queue" on page 58

Notifications of push notification

A message of type PUSH_NOTIFICATION is sent to the system queue when a push notification is received from the server. This message is a notification that messages are queued on the server. It has an expiry of one minute. This expiry time cannot be changed.

This type of system message is useful if you are using the on demand policy. For example, you can call QAManagerBase triggerSendReceive when a system queue message is received of type PUSH_NOTIFICATION.

See also

- "Scenario for messaging with push notifications" on page 6
- "Push notifications" on page 32
- "System queue" on page 58
- "Receiving messages asynchronously" on page 69
- ias_MessageType in "Predefined message properties" on page 659
- "MessageProperties class [QAnywhere .NET]" on page 181
- "MessageProperties class [QAnywhere C++]" on page 356
- "MessageProperties interface [QAnywhere Java]" on page 469

Sending QAnywhere messages

The following procedures describe how to send messages from QAnywhere applications. These procedures assume that you have created and opened a QAManager object.

Sending a message from your application does not ensure it is delivered from your device. It simply places the message on a queue to be delivered. The QAnywhere Agent performs the task of sending the message to the MobiLink server, which in turn delivers it to its destination.

For more information about when message transmission occurs, see "Determining when message transmission should occur on the client" on page 46.

Send a message (.NET)

1. Create a new message.

   You can create either a text message or a binary message, using CreateTextMessage() or CreateBinaryMessage(), respectively.

   ```
   QATextMessage     msg;
   msg = mgr.CreateTextMessage();
   ```

2. Set message properties.

   Use methods of the QATextMessage or QABinaryMessage class to set properties.

   See "QAnywhere messages" on page 13.

3. Put the message on the queue, ready for sending.

   ```
   mgr.PutMessage( "store-id\\queue-name", msg );
   ```

   where *store-id* and *queue-name* are strings that combine to form the destination address.

   See "QAManagerBase.PutMessage method [QAnywhere .NET]" on page 251 and "Determining when message transmission should occur on the client" on page 46.

Send a message (C++)

1. Create a new message.

   You can create either a text message or a binary message, using createTextMessage() or createBinaryMessage(), respectively.

   ```
   QATextMessage *   msg;
   msg = mgr->createTextMessage();
   ```

2. Set message properties.

   Use methods of the QATextMessage or QABinaryMessage class to set message properties.

See "QAnywhere messages" on page 13.

3. Put the message on the queue, ready for sending.

```
if( msg != NULL ) {
  if( !mgr->putMessage( "store-id\\queue-name", msg ) ) {
    // Display error using mgr->getLastErrorMsg().
    }
    mgr->deleteMessage( msg );
}
```

where *store-id* and *queue-name* are strings that combine to form the destination address.

See "QAManagerBase.putMessage method [QAnywhere C++]" on page 416 and "Determining when message transmission should occur on the client" on page 46.

Send a message (Java)

1. Create a new message.

   You can create a text message or a binary message, using QAManagerBase.createTextMessage() or QAManagerBase.createBinaryMessage(), respectively.

```
QATextMessage msg;
msg = mgr.createTextMessage();
```

2. Set message properties.

   Use QATextMessage or QABinaryMessage methods to set message properties.

   See "QAnywhere messages" on page 13.

3. Put the message on the queue.

```
mgr.putMessage("store-id\\queue-name", msg);
```

See "QAManagerBase.putMessage method [QAnywhere Java]" on page 534 and "Determining when message transmission should occur on the client" on page 46.

Send a message (SQL)

1. Declare a variable to hold the message ID.

```
begin
    declare @msgid varchar(128);
```

2. Create a new message.

```
    set @msgid = ml_qa_createmessage();
```

3. Set message properties.

   For more information, see "Message properties" on page 628.

4. Put the message on the queue.

```
        call ml_qa_putmessage( @msgid, 'clientid\queuename' );
        commit;
    end
```

See "ml_qa_putmessage" on page 654 and "Determining when message transmission should occur on the client" on page 46.

Transactional messaging implementation

Transactional messaging provides the ability to group messages in a way that guarantees that either all messages in the group are delivered, or none are. This is more commonly referred to as a single **transaction**.

When implementing transactional messaging, you create a special QAManagerBase object called QATransactionalManager.

See also

- "QATransactionalManager interface [QAnywhere .NET]" on page 295
- "QATransactionalManager class [QAnywhere C++]" on page 457
- "QATransactionalManager interface [QAnywhere Java]" on page 575
- SQL clients: all messaging is transactional for SQL clients and no transactional manager is required

Implementing transactional messaging for .NET clients

Create a transactional manager

1. Initialize QAnywhere.

   This step is the same as in non-transactional messaging.

   ```
   using iAnywhere.QAnywhere.Client;
   ```

2. Create a QATransactionalManager object.

   For example, to create a default QATransactionalManager object, invoke CreateQATransactionalManager:

   ```
   QATransactionalManager mgr = fact.CreateQATransactionalManager();  // no
   argument
   ```

   You can alternatively create a QATransactionalManager object that is customized using a properties file. The properties file is specified in the CreateQATransactionalManager method:

   ```
   QAManagerFactory fact = QAManagerFactory.Instance( "qa_mgr.props" );
   ```

   where *qa_mgr.props* is the name of the properties file that resides on the remote device.

3. Initialize the QAManager object.

```
mgr.Open();
```

You are now ready to send messages. The following procedure sends two messages in a single transaction.

Send multiple messages in a single transaction

1. Initialize message objects.

```
QATextMessage msg_1;
QATextMessage msg_2;
```

2. Send the messages.

   The following code sends two messages in a single transaction:

```
msg_1 = mgr.CreateTextMessage();
msg_2 = mgr.CreateTextMessage();
mgr.PutMessage( "jms_1\\queue_name", msg_1 );
mgr.PutMessage( "jms_1\\queue_name", msg_2 );
mgr.Commit();
```

   The Commit method commits the current transaction and begins a new transaction. This method commits all PutMessage() method and GetMessage() method invocations.

Note
The first transaction begins with the call to open method.

See also

- "QATransactionalManager interface [QAnywhere .NET]" on page 295
- "QAManagerFactory class [QAnywhere .NET]" on page 266

Implementing transactional messaging for C++ clients

Create a transactional manager

1. Initialize QAnywhere.

   This step is the same as in non-transactional messaging.

```
#include <qa.hpp>
QAManagerFactory *   factory;

factory = QAnywhereFactory_init();
if( factory == NULL ) {
    // Fatal error.
}
```

2. Create a transactional manager.


```
QATransactionalManager *  mgr;
mgr = factory->createQATransactionalManager( NULL );
if( mgr == NULL ) {
    // Fatal error.
}
```

As with non-transactional managers, you can specify a properties file to customize QAnywhere behavior. In this example, no properties file is used.

3. Initialize the manager.

```
if( !mgr->open() ) {
   // Display message using mgr->getLastErrorMsg().
}
```

You are now ready to send messages. The following procedure sends two messages in a single transaction.

Send multiple messages in a single transaction

1. Initialize message objects.

```
QATextMessage *  msg_1;
QATextMessage *  msg_2;
```

2. Send the messages.

   The following code sends two messages in a single transaction:

```
msg_1 = mgr->createTextMessage();
if( msg_1 != NULL ) {
  msg_2 = mgr->createTextMessage();
  if( msg_2 != NULL ) {
    if( !mgr->putMessage( "jms_1\\queue_name", msg_1 ) ) {
      // Display message using mgr->getLastErrorMsg().
    } else {
      if( !mgr->putMessage( "jms_1\\queue_name", msg_2 ) ) {
        // Display message using mgr->getLastErrorMsg().
      } else {
        mgr->commit();
      }
    }
    mgr->deleteMessage( msg_2 );
  }
  mgr->deleteMessage( msg_1 );
}
```

The commit method commits the current transaction and begins a new transaction. This method commits all putMessage() method and getMessage() method invocations.

Note
The first transaction begins with the call to open method.

See also

- “QATransactionalManager class [QAnywhere C++]” on page 457
- “QATransactionalManager interface [QAnywhere Java]” on page 575

Implementing transactional messaging for Java clients

Create a transactional manager

1. Initialize QAnywhere.

   This step is the same as in non-transactional messaging.

   ```
   import ianywhere.qanywhere.client.*;
   QAManagerFactory fact = QAManagerFactory.getInstance();
   ```

2. Create a QATransactionalManager object.

   For example, to create a default QATransactionalManager object, invoke createQATransactionalManager with null as its parameter:

   ```
   QATransactionalManager mgr = fact.createQATransactionalManager();  // no
   argument
   ```

   You can alternatively create a QATransactionalManager object that is customized using a properties file. The properties file is specified in the createQATransactionalManager method:

   ```
   mgr = factory.createQATransactionalManager( "qa_mgr.props" );
   ```

   where *qa_mgr.props* is the name of the properties file that resides on the remote device.

3. Initialize the QAManager object.

   ```
   mgr.open();
   ```

You are now ready to send messages. The following procedure sends two messages in a single transaction.

Send multiple messages in a single transaction

1. Initialize message objects.

   ```
   QATextMessage msg_1;
   QATextMessage msg_2;
   ```

2. Send the messages.

   The following code sends two messages in a single transaction:

   ```
   msg_1 = mgr.createTextMessage();
   msg_2 = mgr.createTextMessage();
   mgr.putMessage( "jms_1\\queue_name", msg_1 );
   mgr.putMessage( "jms_1\\queue_name", msg_2 );
   mgr.commit();
   ```

   The commit method commits the current transaction and begins a new transaction. This method commits all putMessage() method and getMessage() method invocations.

Note
The first transaction begins with the call to open method.

See also

- "QAManagerFactory class [QAnywhere Java]" on page 545

Canceling QAnywhere messages

Canceling a QAnywhere message puts the message into a canceled state before it is transmitted. With the default delete rules of the QAnywhere Agent, canceled messages are eventually deleted from the message store. Canceling a QAnywhere message fails if the message is already in a final state, or if it has been transmitted to the central messaging server.

The following procedures describe how to cancel QAnywhere messages.

Note
You cannot cancel a message using the QAnywhere SQL API.

Cancel a message (.NET)

1. Get the ID of the message to cancel.

```
// msg is a QAMessage instance that has not been
// transmitted.
string msgID = msg.getMessageID();
```

2. Call CancelMessage with the ID of the message to cancel.

```
mgr.CancelMessage(msgID);
```

Cancel a message (C++)

1. Get the ID of the message to cancel.

```
// msg is a QAMessage instance that has not been
// transmitted.
qa_string msgID = msg->getMessageID();
```

2. Call cancelMessage with the ID of the message to cancel.

```
bool result = mgr->cancelMessage(msgID);
```

Cancel a message (Java)

1. Get the ID of the message to cancel.

```
// msg is a QAMessage instance that has not been
// transmitted.
String msgID = msg.getMessageID();
```

2. Call cancelMessage with the ID of the message to cancel.

```
boolean result = mgr.cancelMessage(msgID);
```

See also

- "QAManagerBase.CancelMessage method [QAnywhere .NET]" on page 235
- "QAManagerBase.cancelMessage method [QAnywhere C++]" on page 402
- "QAManagerBase.cancelMessage method [QAnywhere Java]" on page 520

Receive QAnywhere messages

The following topics describe how to receive QAnywhere messages.

Receiving messages synchronously

To receive messages synchronously, your application explicitly polls the queue for messages. It may poll the queue periodically, or when a user initiates a particular action such as clicking a Refresh button.

Receive messages synchronously (.NET)

1. Declare message objects to hold the incoming messages.

```
QAMessage msg;
QATextMessage text_msg;
```

2. Poll the message queue, collecting messages:

```
for(;;) {
    msg = mgr.GetMessageNoWait("queue-name");
    if( msg == null ) {
      break;
    }
    addMessage( msg );
  }
```

Receive messages synchronously (C++)

1. Declare message objects to hold the incoming messages.

```
QAMessage * msg;
QATextMessage * text_msg;
```

2. Poll the message queue, collecting messages:

```
for( ;; ) {
    msg = mgr->getMessageNoWait( "queue-name" );
    if( msg == NULL ) {
```



```
        break;
      }
      addMessage(msg);
    }
```

Receive messages synchronously (Java)

1. Declare message objects to hold the incoming messages.

```
QAMessage msg;
QATextMessage text_message;
```

2. Poll the message queue, collecting messages:

```
if(mgr.start()) {
  for ( ;; ) {
    msg = mgr.getMessageNoWait("queue-name");
    if ( msg == null ) {
      break;
    }
    addMessage(msg);
  }
  mgr.stop();
}
```

Receive messages synchronously (SQL)

1. Declare an object to hold the message ID.

```
begin
    declare @msgid varchar(128);
```

2. Poll the message queue, collecting messages.

```
    loop
        set @msgid = ml_qa_getmessagenowait( 'myaddress' );
        if @msgid is null then leave end if;
        message 'a message with content ' ||
ml_qa_gettextcontent( @msgid ) || ' has been received';
    end loop;
    commit;
end
```

See:

- “QAManagerBase.GetMessageNoWait method [QAnywhere .NET]” on page 244
- “QAManagerBase.getMessageNoWait method [QAnywhere C++]” on page 412
- “QAManagerBase.getMessageNoWait method [QAnywhere Java]” on page 528
- “ml_qa_getmessagenowait” on page 650
- “ml_qa_getmessagetimeout” on page 651
- “ml_qa_getmessage” on page 649

Receiving messages asynchronously

To receive messages asynchronously using the .NET, C++, and Java APIs, you can write and register a message listener function that is called by QAnywhere when a message appears in the queue. The

message listener takes the incoming message as a parameter. The task you perform in your message listener depends on your application. For example, in the TestMessage sample application the message listener adds the message to the list of messages in the main TestMessage window.

Development tip for .NET, C++ and Java

It is safer to use QAManagers in mode EXPLICIT_ACKNOWLEDGEMENT to guard against the possibility of an application error occurring part way through the processing of received messages and the message being acknowledged anyway.

If the QAManager is opened in mode EXPLICIT_ACKNOWLEDGEMENT, the message can be acknowledged in the onMessage method only after it has been successfully processed. That way if there was an error processing the message, the message is received again because it was not acknowledged.

If the QAManager is opened in mode IMPLICIT_ACKNOWLEDGEMENT, the message passed to onMessage is acknowledged implicitly when onMessage returns. If the user application encounters an error while processing the message, the message is acknowledged and never received again.

Receive messages asynchronously (.NET)

1. Implement a message handler method.

```
private void onMessage(QAMessage msg) {
    // Process message.
}
```

2. Register the message handler.

   To register a message handler, create a QAManager.MessageListener object that has the message handler function as its argument. Then use the QAManager.SetMessageListener function to register the MessageListener with a specific queue. In the following example, *queue-name* is a string that is the name of the queue the QAManager object listens to.

```
MessageListener listener;
listener = new MessageListener( onMessage );
mgr.SetMessageListener( "queue-name", listener );
```

Receive messages asynchronously (C++)

1. Create a class that implements the QAMessageListener interface.

```
class MyClass: public QAMessageListener {
   public:
   void onMessage( QAMessage * Msg);
};
```

2. Implement the onMessage method.

   The QAMessageListener interface contains one method, onMessage. Each time a message arrives in the queue, the QAnywhere library calls this method, passing the new message as the single argument.

```
void MyClass::onMessage(QAMessage * msg) {
     // Process message.
}
```


3. Register the message listener.

```
my_listener = new MyClass();
mgr->setMessageListener( "queue-name", my_listener );
```

Receive a message asynchronously (Java)

1. Implement a message handler method and an exception handler method.

```
class MyClass implements QAMessageListener {
  public void onMessage(QAMessage message) {
    // Process the message.
  }
  public void onException(
    QAException exception, QAMessage message) {
    // Handle the exception.
  }
}
```

2. Register the message handler.

```
MyClass listener = new MyClass();
mgr.setMessageListener("queue-name", listener);
```

Receive messages asynchronously (SQL)

- Create a stored procedure with the name **ml_qa_listener_*queue***, where *queue* is the name of a message queue.

  This procedure is called whenever a message is queued on the given queue.

See also

- “MessageListener delegate [QAnywhere .NET]” on page 301
- “QAManagerBase.SetMessageListener method [QAnywhere .NET]” on page 257
- “QAMessageListener class [QAnywhere C++]” on page 451
- “QAManagerBase.setMessageListener method [QAnywhere C++]” on page 420
- “QAMessageListener interface [QAnywhere Java]” on page 566
- “QAManagerBase.setMessageListener method [QAnywhere Java]” on page 539
- “ml_qa_listener_queue” on page 653

Receiving messages using a selector

You can use **message selectors** to select messages for receiving. A message selector is a SQL-like expression that specifies a condition to select a subset of messages to consider for receive operations.

The syntax and semantics of message selectors are exactly the same as the condition part of transmission rules.

See “Condition syntax” on page 731.

Example

Note
The SQL API does not support receiving messages using a selector.

The following C# example gets the next message from receiveQueue that has a message property called intprop with value 1.

```
msg = receiver.GetMessageBySelectorNoWait(
   receiveQueue, "intprop=1" );
```

The following C++ example gets the next message from receiveQueue that has a message property called intprop with value 1.

```
msg = receiver->getMessageBySelectorNoWait(
  receiveQueue, "intprop=1" );
```

The following Java example gets the next message from receiveQueue that has a message property called intprop with value 1.

```
msg = receiver.getMessageBySelectorNoWait(
  receiveQueue, "intprop=1");
```

See also

- “QAManagerBase.GetMessageBySelector method [QAnywhere .NET]” on page 242
- “QAManagerBase.GetMessageBySelectorNoWait method [QAnywhere .NET]” on page 242
- “QAManagerBase.getMessageBySelector method [QAnywhere C++]” on page 410
- “QAManagerBase.getMessageBySelectorNoWait method [QAnywhere C++]” on page 411
- “QAManagerBase.getMessageBySelector method [QAnywhere Java]” on page 526
- “QAManagerBase.getMessageBySelectorNoWait method [QAnywhere Java]” on page 526

Very large messages

Sometimes messages are so large that they exceed the limit set with the QAManager property MAX_IN_MEMORY_MESSAGE_SIZE or its defaults of 1MB on Windows and 64KB on Windows Mobile. In this case, the message object cannot contain the full content of the message in memory, so methods that rely on the full content of the message being loaded into memory, such as readInt() and readString(), cannot be used. However, you can read very large messages directly from the message store in pieces. To do this, use QATextMessage.readText() or QABinaryMessage.readBinary() in a loop.

Note
The SQL API does not support receiving very large messages.

For more information, see:

When you do this, you cannot use a QAManager that was opened with IMPLICIT_ACKNOWLEDGEMENT. You must use a QAManager that was opened with EXPLICIT_ACKNOWLEDGEMENT and you must complete all calls to readText() or readBinary() before acknowledging the message.

See also

- "QABinaryMessage.ReadBinary method [QAnywhere .NET]" on page 194
- "QATextMessage.ReadText method [QAnywhere .NET]" on page 293
- "QABinaryMessage.readBinary method [QAnywhere C++]" on page 372
- "QATextMessage.readText method [QAnywhere C++]" on page 456
- "QABinaryMessage.readBinary method [QAnywhere Java]" on page 486
- "QATextMessage.readText method [QAnywhere Java]" on page 572
- "Acknowledgement modes" on page 57

QAnywhere message browsing

You can browse messages in incoming and outgoing queues. Browse operations do not affect the status of messages.

The following topics describe how to browse QAnywhere messages.

Browse all messages

You can browse the messages in all queues by calling the appropriate browseMessages() method.

The following .NET example uses the QAManager.BrowseMessages() method to browse all queues:

```
QAMessage msg;
IEnumerator msgs = mgr.BrowseMessages();
while( msgs.MoveNext() ) {
      msg = (QAMessage)msgs.Current;
      // Process message.
}
```

The following C++ example uses the QAManager browseMessages function to browse all queues:

```
QAMessage *msg;
qa_browse_handle bh = mgr->browseMessages();
for (;;) {
  msg = mgr->browseNextMessage( bh );
  if( msg == qa_null ) {
    break;
  }
  // Process message.
  mgr->browseClose( bh );
}
```

The following Java example uses the QAManager.browseMessages method to browse all queues:

```
QAMessage msg;
java.util.Enumeration msgs = mgr.browseMessages();
while( msgs.hasMoreElements() ) {
    msg = (QAMessage)msgs.nextElement();
  // Process message.
}
```

See also

- "Predefined message properties" on page 659
- "QAManagerBase.BrowseMessages method [QAnywhere .NET]" on page 232
- "QAManagerBase.browseMessages method [QAnywhere C++]" on page 399
- "QAManagerBase.browseMessages method [QAnywhere Java]" on page 517
- SQL: The SQL API does not support browsing messages

Browsing messages in a queue

You can browse the messages in a given queue by supplying the queue name to the appropriate browseMessagesByQueue() method.

The following .NET example uses the QAManager.BrowseMessagesByQueue method to browse a queue:

```
QAMessage msg;
IEnumerator msgs = mgr.BrowseMessagesByQueue( "q1" );
while( msgs.MoveNext() ) {
  msg = (QAMessage)msgs.Current;
  // Process message.
}
```

The following C++ example uses the QAManager browseMessagesByQueue function to browse a queue:

```
QAMessage *msg;
qa_browse_handle bh = mgr->browseMessagesByQueue( _T("q1") );
for (;;) {
  msg = mgr->browseNextMessage( bh );
  if( msg == qa_null ) {
    break;
  }
  // Process message.
}
mgr->browseClose( bh );
```

The following Java example uses the QAManager.browseMessagesByQueue method to browse a queue:

```
QAMessage msg;
java.util.Enumeration msgs = mgr.browseMessagesByQueue( "q1" );
while( msgs.hasMoreElements() ) {
  msg = (QAMessage)msgs.nextElement();
  // Process message.
}
```

See also

- "QAManagerBase.BrowseMessagesByQueue method [QAnywhere .NET]" on page 234
- "QAManagerBase.browseMessagesByQueue method [QAnywhere C++]" on page 400
- "QAManagerBase.browseMessagesByQueue method [QAnywhere Java]" on page 518
- SQL: The SQL API does not support browsing messages

Browsing a message by ID

You can browse a particular message by specifying its ID to a browseMessagesbyID() method.

The following .NET example uses the QAManager.BrowseMessageByID method to browse a message:

```
QAMessage msg;
IEnumerator msgs = mgr.BrowseMessagesByID( "ID:123" );
if( msgs.MoveNext() ) {
  msg = (QAMessage)msgs.Current;
  // Process message.
}
```

The following C++ example uses the QAManager browseMessageByID function to browse a message:

```
QAMessage *msg;
qa_browse_handle bh = mgr->browseMessagesByID( _T( "ID:123" ) );
msg = mgr->browseNextMessage( bh );
if( msg != qa_null ) {
  // Process message.
}
mgr->browseClose( bh );
```

The following Java example uses the QAManager.browseMessageByID method to browse a message:

```
QAMessage msg;
java.util.Enumeration msgs = mgr.browseMessagesByID( "ID:123" );
if( msgs.hasMoreElements() ) {
  msg = (QAMessage)msgs.nextElement();
  // Process message.
}
```

See also

- "QAManagerBase.BrowseMessagesByID method [QAnywhere .NET]" on page 233
- "QAManagerBase.browseMessagesByID method [QAnywhere C++]" on page 400
- "QAManagerBase.browseMessagesByID method [QAnywhere Java]" on page 518
- SQL: The SQL API does not support browsing messages

Browsing messages using a selector

You can use **message selectors** to select messages for browsing. A message selector is a SQL-like expression that specifies a condition to select a subset of messages to consider for browse operations.

The syntax and semantics of message selectors are exactly the same as the condition part of transmission rules.

The following .NET example browses all messages in the message store that have a property called intprop with value 1.

```
QAMessage msg;
IEnumerator msgs = mgr.BrowseMessagesBySelector( "intprop = 1" );
while( msgs.MoveNext() ) {
  msg = (QAMessage)msgs.Current;
  // Process message.
}
```

The following C++ example browses all messages in the message store that have a property called intprop with value 1.

```
QAMessage *msg;
qa_browse_handle bh = mgr->browseMessagesBySelector( _T("intprop = 1") );
for (;;) {
  msg = mgr->browseNextMessage( bh );
  if( msg == qa_null ) {
    break;
  }
  // Process message.
}
mgr->browseClose( bh );
```

The following Java example browses all messages in the message store that have a property called intprop with value 1.

```
QAMessage msg;
java.util.Enumeration msgs = mgr.browseMessagesBySelector( "intprop = 1" );
while( msgs.hasMoreElements() ) {
  msg = (QAMessage)msgs.nextElement();
  // Process message.
}
```

See also

- "Condition syntax" on page 731
- "QAManagerBase.BrowseMessagesBySelector method [QAnywhere .NET]" on page 235
- "QAManagerBase.browseMessagesBySelector method [QAnywhere C++]" on page 401
- "QAManagerBase.browseMessagesBySelector method [QAnywhere Java]" on page 519
- SQL: The SQL API does not support browsing messages

QAnywhere exception handling

The QAnywhere C++, Java, and .NET APIs include special objects and properties for exception handling.

.NET exceptions

The QAException class encapsulates QAnywhere client application exceptions. After you catch a QAnywhere exception, you can use the QAException ErrorCode and Message properties to determine the error code and error message.

Note that if a QAException is thrown inside a message listener delegate and it is not caught in the message listener, then it gets logged to the QAManager log file. Since uncaught QAExceptions are only logged, it is recommended that all exceptions be handled within message listener delegates or handled by exception listener delegates so that they can be dealt with appropriately.

For more information about message listener delegates and exception listener delegates, see:

- "MessageListener delegate [QAnywhere .NET]" on page 301
- "MessageListener2 delegate [QAnywhere .NET]" on page 301
- "ExceptionListener delegate [QAnywhere .NET]" on page 300
- "ExceptionListener2 delegate [QAnywhere .NET]" on page 300

For more information about the log file, see "QAnywhere manager configuration properties" on page 80.

When a QAException is thrown, the current transaction is rolled back. When this happens in a message listener with a QATransactionalManager, the message that was being processed when the QAException was thrown is put back in the receive queue and so that it can be re-received. You can use the message store property ias_MaxDeliveryAttempts to prevent an infinite loop.

When the property ias_MaxDeliveryAttempts is set to a positive integer *n* by a QAnywhere application, as in `mgr.SetIntStoreProperty( "ias_MaxDeliveryAttempts", 5 )`, the QAnywhere client attempts to receive an unacknowledged message up to *n* times before setting the status of the message to unreceivable. If the property ias_MaxDeliveryAttempts is not set or is negative, the QAnywhere client attempts to receive messages an unlimited number of times.

For more information, see:

- "QAException class [QAnywhere .NET]" on page 209
- "QAException.ErrorCode property [QAnywhere .NET]" on page 212
- "Predefined client message store properties" on page 717

C++ exceptions

For C++, the QAError class encapsulates QAnywhere client application exceptions. You can use the QAManagerBase::getLastError() method or QAManagerFactory::getLastError() method to determine the error code associated with the last executed method. You can use the corresponding getLastErrorMessage() method to obtain the error text.

For a list of error codes and more information, see "QAError class [QAnywhere C++]" on page 380.

For more information about getLastError and getLastErrorMessage, see:

- "QAManagerBase.getLastError method [QAnywhere C++]" on page 408
- "QAManagerBase.getLastErrorMsg method [QAnywhere C++]" on page 408
- "QAManagerFactory.getLastError method [QAnywhere C++]" on page 427
- "QAManagerFactory.getLastErrorMsg method [QAnywhere C++]" on page 428

Java exceptions

The QAException class encapsulates QAnywhere client application exceptions. After you catch a QAnywhere exception, you can use the QAException ErrorCode and Message properties to determine the error code and error message.

If a QAException is thrown inside a message listener and it is not caught in the message listener, then it is logged to the QAManager log file. Since uncaught QAExceptions are only logged, it is recommended that all exceptions be handled within message listeners or handled by exception listeners so that they can be dealt with appropriately.

For more information about message listeners and exception listeners, see:

- "QAMessageListener interface [QAnywhere Java]" on page 566
- "QAMessageListener2 interface [QAnywhere Java]" on page 568
- "QAException class [QAnywhere Java]" on page 498

For more information about the log file, see "QAnywhere manager configuration properties" on page 80.

When a QAException is thrown, the current transaction is rolled back. When this happens in a message listener with a QATransactionalManager, the message that was being processed when the QAException was thrown is put back in the receive queue and so that it can be re-received. You can use the message store property ias_MaxDeliveryAttempts to prevent an infinite loop.

When the property ias_MaxDeliveryAttempts is set to a positive integer *n* by a QAnywhere application, as in `mgr.SetIntStoreProperty( "ias_MaxDeliveryAttempts", 5 )`, the QAnywhere client attempts to receive an unacknowledged message up to *n* times before setting the status of the message to unreceivable. If the property ias_MaxDeliveryAttempts is not set or is negative, the QAnywhere client attempts to receive messages an unlimited number of times.

For more information, see:

- "QAException.ErrorCode property [QAnywhere .NET]" on page 212
- "Predefined client message store properties" on page 717

Error codes

The following table lists QAnywhere error code values:

Error value	Description
0	No error.
1000	Initialization error.
1001	Termination error.
1002	Unable to access the client properties file.
1003	No destination.
1004	The function is not implemented.
1005	You cannot write to a message as it is in read-only mode.
1006	Error storing a message in the client message store.
1007	Error retrieving a message from the client message store.
1008	Error initializing the background thread.
1009	Error opening a connection to the message store.
1010	There is an invalid property in the client properties file.
1011	Error opening the log file.
1012	Unexpected end of message reached.

Error value	Description
1013	The message store is too large relative to the free disk space on the device.
1014	The message store has not been initialized for messaging.
1015	Error getting queue depth.
1016	Cannot use QAManagerBase.getQueueDepth when the message store ID has not been set.
1017	Cannot use QAManagerBase.getQueueDepth on a given destination when filter is ALL.
1018	Error canceling message.
1019	Error canceling message. Cannot cancel a message that has already been sent.
1020	Error acknowledging the message.
1021	The QAManager is not open.
1022	The QAManager is already open.
1023	The given selector has a syntax error.
1024	The timestamp is outside the acceptable range.
1025	Cannot open QAManager because the maximum number of concurrent server requests is not high enough. See "-gn dbsrv12 server option" [*SQL Anywhere Server - Database Administration*].
1026	Error retrieving property from message store.
1027	Error storing property to message store.

Shutting down QAnywhere

After you have completed sending and receiving messages, you can shut down the QAnywhere messaging system by completing one of the following procedures.

Shut down QAnywhere (.NET)

- Stop and close the QAnywhere manager.

```
mgr.Stop();
mgr.Close();
```

Shut down QAnywhere (C++)

1. Close the QAnywhere manager.

   ```
   mgr->stop();
   mgr->close();
   ```

2. Terminate the factory.

   ```
   QAnywhereFactory_term();
   ```

   This step shuts down the messaging part of your application.

Shut down QAnywhere (Java)

- Stop and close the QAnywhere manager.

  ```
  mgr.stop();
  mgr.close();
  ```

See also

- "QAManagerBase.Stop method [QAnywhere .NET]" on page 265
- "QAManagerBase.stop method [QAnywhere C++]" on page 424
- "QAManagerBase.stop method [QAnywhere Java]" on page 544
- SQL: The SQL API does not support shutting down QAnywhere

Multi-threading considerations

Access to a QAManager is serialized. When you have multiple threads accessing a single QAManager, threads block while one thread performs a method call on the QAManager. To maximize concurrency, use a different QAManager for each thread. Only one thread is allowed to access an instance of QAManager at one time. Other threads block until the QAManager method that was invoked by the first thread returns.

QAnywhere manager configuration properties

You can set QAnywhere manager configuration properties in one of the following ways:

- Create a properties text file to define the QAnywhere manager configuration properties that is used by one Manager instance.

  See "QAnywhere manager configuration property settings in a file" on page 82.

- Set QAnywhere manager configuration properties programmatically.

  See "Setting QAnywhere manager configuration properties programmatically" on page 83.

The following are the QAnywhere manager configuration properties:

- **COMPRESSION_LEVEL=n** Set the compression level.

n is the compression factor, which is expressed as is an integer between 0 and 9, where 0 indicates no compression and 9 indicates maximum compression.

- **CONNECT_PARAMS=connect-string** Specify a connection string for the QAnywhere manager to use to connect to the message store database. Specify each connection option in the form *keyword=value* with multiple options separated by semicolons.

  This property is not supported in the standalone client.

  The default is `"server=qanywhere;uid=ml_qa_user;pwd=qanywhere"`.

  For a list of options, see "Connection parameters" [*SQL Anywhere Server - Database Administration*].

  For information about managing the database user and password, see "Secure messaging applications" on page 115.

- **DATABASE_TYPE=string** Specify the type of database the QAnywhere manager is connected to. Use **sqlanywhere** for a SQL database, or **ultralite** for an UltraLite database. By default, the manager uses **sqlanywhere**.

- **LOG_FILE=filename** Specify the name of a file to use to write logging messages. Specifying this option implicitly enables logging.

- **MAX_IN_MEMORY_MESSAGE_SIZE=n** When reading a message, *n* is the largest message, in bytes, for which a buffer is allocated. A message larger than *n* bytes must be read using streaming operations. The default value is 1MB on Windows and 64KB on Windows Mobile.

The following are properties made exclusively for the standalone client:

- **ML_PROTOCOL_TYPE** Specify the protocol type. Valid options are **tcpip**, **tls**, **http**, or **https**.

- **ML_PROTOCOL_PARAMS** Specify the MobiLink connect parameters.

- **ML_PROTOCOL_USENAME** Performs the same effect as the -mu option in QAnywhere agent.

- **ML_PROTOCOL_PASSWORD** Performs the same effect as the -mn option in QAnywhere agent.

- **INC_UPLOAD** Performs the same effect as the -iu option in QAnywhere agent.

- **INC_DOWNLOAD** Performs the same effect as the -idl option in QAnywhere agent.

- **STORE_ID** Performs the same effect as the -id option in QAnywhere agent.

- **STORE_ENCRYPTION_KEY** Specify the encryption key to encrypt the MessageStore.

- **POLICY** Performs the same effect as the -policy option in QAnywhere agent.

- **DELETE_PERIOD** Specifies the number of seconds between executions of the delete rules. If the amount specified is a negative number, execution of the delete rules is disabled.

- **PUSH** Performs the same effect as the -push option in the QAnywhere agent.

QAnywhere manager configuration property settings in a file

Note
You can create or open a QAnywhere manager configuration file in Sybase Central. From the QAnywhere 12 plug-in task pane, click **Create An Agent Configuration File**. When you have chosen a file name and location, the **Properties** window for the configuration file opens, where you can set the properties.

The information in a QAnywhere manager properties file is specific to one instance of a QAManager.

When using a properties file, it must be configured for and installed on the remote device with each deployed copy of your application.

For information about specifying the name of the property file, see:

- "QAManagerFactory.CreateQAManager method [QAnywhere .NET]" on page 267
- "QAManagerFactory.createQAManager method [QAnywhere C++]" on page 425
- "QAManagerFactory.createQAManager method [QAnywhere Java]" on page 546
- SQL API: You cannot set properties in a file using the QAnywhere SQL API. See "Setting QAnywhere manager configuration properties programmatically" on page 83.

If the properties file does not reside in the same directory as your client executable, you must also specify the absolute path. If you want to use the default settings for the properties, use null instead of a file name.

Values set in the file permit you to enable or disable some of the QAnywhere features, such as automatic message compression and logging.

Entries in a QAnywhere manager configuration properties file take the form *name=value*. If *value* has spaces, enclose it in double quotes. Comment lines start with **#**. For example:

```
# contents of QAnywhere manager configuration properties file
LOG_FILE=.\sender.ini.txt
# A comment
CONNECT_PARAMS=server=qanywhere;uid=ml_qa_user;pwd=qanywhere
DATABASE_TYPE=sqlanywhere
MAX_IN_MEMORY_MESSAGE_SIZE=2048
COMPRESSION_LEVEL=0
```

For a list of property names, see "QAnywhere manager configuration properties" on page 80.

Referencing the configuration file

Suppose you have a QAnywhere manager configuration properties file called *mymanager.props* with the following content:

```
COMPRESSION_LEVEL=9
CONNECT_PARMS=DBF=mystore.db
```

When you create QAManager, you reference the file by name.

The following is an example using C#:


```
QAManager     mgr;
mgr = QAManagerFactory.Instance.CreateQAManager( "mymanager.props" );
mgr.Open( AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT );
```

For the .NET API, see "QAManager interface [QAnywhere .NET]" on page 221 and "QAManagerFactory class [QAnywhere .NET]" on page 266.

The following is an example using C++:

```
QAManagerFactory *  qa_factory;
QAManager *    mgr;
qa_factory = QAnywhereFactory_init();
mgr = qa_factory->createQAManager( "mymanager.props" );
mgr->open( AcknowledgementMode::EXPLICIT_ACKNOWLEDGEMENT );
```

For the C++ API, see "QAManager class [QAnywhere C++]" on page 388 and "QAManagerFactory class [QAnywhere C++]" on page 425.

The following is an example using Java:

```
QAManager mgr;
mgr = QAManagerFactory.getInstance().createQAManager("mymanager.props");
mgr.open( AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT );
```

For the Java API, see "QAManagerFactory class [QAnywhere Java]" on page 545 and "QAManager interface [QAnywhere Java]" on page 507.

Setting QAnywhere manager configuration properties programmatically

In the QAnywhere APIs, you can use the QAManagerBase set property method to set properties programmatically. Setting QAnywhere manager configuration properties programmatically must be done before calling the open method of a QAManager instance.

Example

The following C# example sets properties programmatically. When you create the QAManager, you specify the property settings.

```
QAManager     mgr;
mgr = QAManagerFactory.Instance.CreateQAManager( "null" );
mgr.SetProperty( "COMPRESSION_LEVEL", "9" );
mgr.SetProperty( "CONNECT_PARAMS", "DBF=mystore.db" );
mgr.SetProperty( "DATABASE_TYPE", "sqlanywhere" );
mgr.Open( AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT );
```

The following C++ example sets properties programmatically. When you create the QAManager, you specify the property settings.

```
QAManagerFactory *  qa_factory;
QAManager *    mgr;
qa_factory = QAnywhereFactory_init();
mgr = qa_factory->createQAManager( NULL );
mgr->setProperty( "COMPRESSION_LEVEL", "9" );
```

```
mgr->setProperty( "CONNECT_PARAMS", "DBF=mystore.db" );
mgr->setProperty( "DATABASE_TYPE", "sqlanywhere" );
mgr->open( AcknowledgementMode::EXPLICIT_ACKNOWLEDGEMENT );
```

The following Java example sets properties programmatically. When you create the QAManager, you specify the property settings.

```
QAManager mgr;
mgr = QAManagerFactory.getInstance().createQAManager();
mgr.setProperty("COMPRESSION_LEVEL", "9");
mgr.setProperty("CONNECT_PARMS", "DBF=mystore.db");
mgr.setProperty("DATABASE_TYPE", "sqlanywhere");
mgr.open(AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT);
```

See also

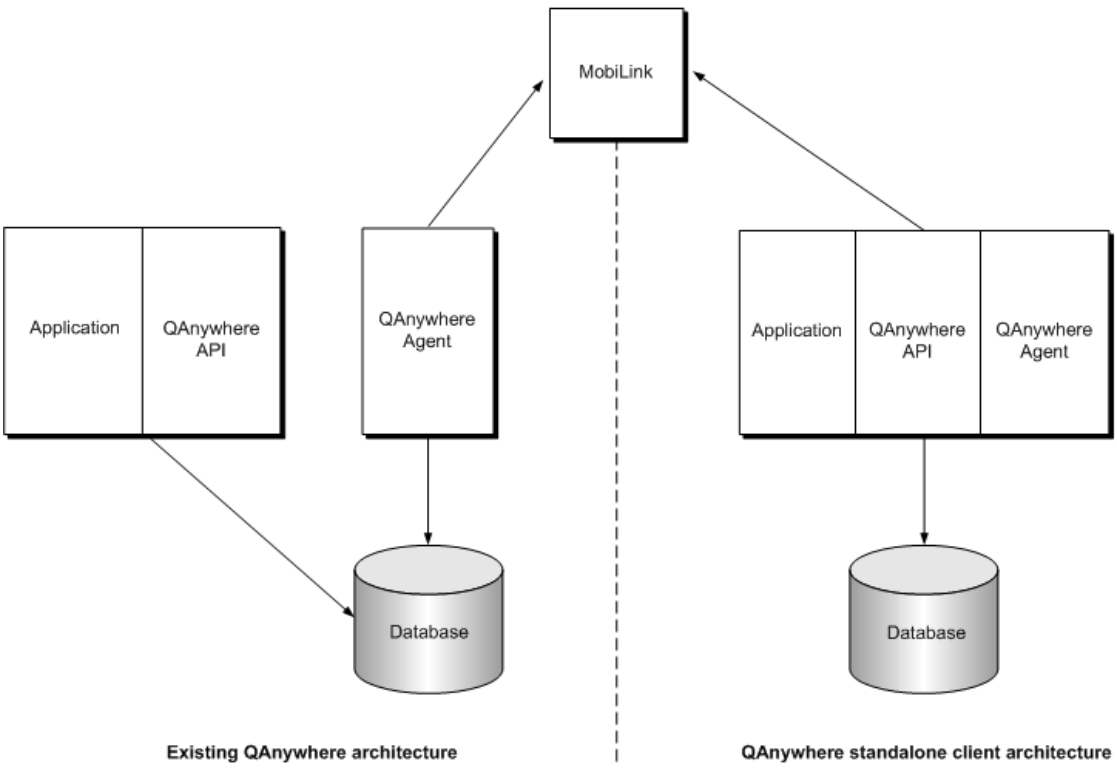
- “QAnywhere manager configuration properties” on page 80
- “QAManager interface [QAnywhere .NET]” on page 221
- “QAManagerFactory class [QAnywhere .NET]” on page 266
- “QAManager class [QAnywhere C++]” on page 388
- “QAManagerFactory class [QAnywhere C++]” on page 425
- “QAManagerFactory class [QAnywhere Java]” on page 545
- “QAManager interface [QAnywhere Java]” on page 507

QAnywhere standalone client

The QAnywhere standalone client provides a compact client that enables you to set up a messaging system without worrying about running the QAnywhere Agent or administering your database. The standalone client incorporates both client store administration and QAnywhere Agent functionality into the same process that accesses the client API, so you are no longer required to create or maintain the client message stores and do not need to run the QAnywhere Agent as a separate process.

The following diagram shows the standalone client architecture and the existing QAnywhere architecture:

Existing QAnywhere architecture versus QAnywhere standalone client architecture



The standalone client message store

The message store is an UltraLite database that is bound to a store ID and is created and maintained automatically by the standalone client. The QAnywhere API accesses the message store using the in-process UltraLite runtime and not the UltraLite engine.

The standalone client message store differs from existing QAnywhere message stores in that only a single user application can access a message store at a time. That is, multiple standalone client applications on the same device require separate message stores and distinct store IDs. As a result of this, the Standalone

Client has no notion of "local messaging" whereby messages can be sent to different queues in the same message store. All messages sent through the standalone client are assumed to be for a different message store.

Since messages may reside in the message store even while the client application is not running, you can secure the store by providing a STORE_ENCRYPTION_KEY value on first use. The same key must then be provided each subsequent time a user attempts to use the message store. This encryption key also encrypts the data on disc.

See also

- "QAnywhere manager configuration properties" on page 80

Standalone client deployment

The QAnywhere standalone client is distributed in a separate dll file for .NET and a separate JAR file for Java:

- The .NET implementation is distributed in the dll *iAnywhere.QAnywhere.StandAloneClient.dll* and uses namespace iAnywhere.QAnywhere.StandAloneClient
- The Java implementation is distributed in the JAR file *qastandaloneclient.jar* and uses the package name ianywhere.qanywhere.standaloneclient.

See also

- "QAnywhere .NET API reference for clients" on page 181
- "QAnywhere Java API reference for clients" on page 468

Standalone client API

Both the Java and .NET implementations of the standalone client preserve the same API as the existing clients with the following exceptions.

The following QAManager configuration properties are exclusively for the standalone client:

- **ML_PROTOCOL_TYPE** Specifies the protocol type. Valid options are tcpip, tls, http, or https.
- **ML_PROTOCOL_PARAMS** Specifies MobiLink connection parameters.
- **ML_PROTOCOL_USENAME** Specifies the MobiLink user name. This is the same as the QAnywhere Agent -mu option. See "-mu qaagent option" on page 682.
- **ML_PROTOCOL_PASSWORD** Specifies a new password for the MobiLink user. This is the same as the QAnywhere Agent -mn option. See "-mn qaagent option" on page 681.
- **INC_UPLOAD** Specifies the incremental upload size. This is the same as the QAnywhere Agent -iu option. See "-iu qaagent option" on page 680.

- **INC_DOWNLOAD** Specifies the incremental download size. This is the same as the QAnywhere Agent -idl option. See "-idl qaagent option" on page 679.
- **STORE_ID** Specifies the ID of the client message store that the standalone client is to connect to. This is the same as the QAnywhere Agent -id option. See "-id qaagent option" on page 678.
- **STORE_ENCRYPTION_KEY** Specifies the encryption key used to encrypt the message store.
- **POLICY** Specifies a policy that determines when message transmission occurs. This is the same as the QAnywhere Agent -policy option. See "-policy qaagent option" on page 685.
- **DELETE_PERIOD** Specifies the number of seconds between execution of deletion of messages that have reached a final state.
- **PUSH** Specifies how push notifications are delivered. This is the same as the QAnywhere Agent -push option, except that the default is `-push lwpoll` for the standalone client. See "-push qaagent option" on page 687.

The following QAManager configuration properties are not supported in the QAnywhere standalone client:

- CONNECT_PARAMS
- DATABASE_TYPE

See also

- "QAnywhere manager configuration properties" on page 80
- "qaagent utility" on page 672

Mobile web services

Web Services have become a popular way to expose application functionality and enable better interoperability between the resources of various enterprises. They broaden the capabilities of mobile applications and simplify the development process.

Implementing web services in a mobile environment can be challenging because connectivity may not be available (or may be interrupted) and because of other limitations of wireless environments and devices. For example, a user working with a mobile application may want to make a request to a web service while offline and obtain the response when they go online, or an IT administrator may want to specify rules that restrict the size of web service responses based on the type of network connectivity the mobile application is using (such as GPRS, 802.11, or cradled).

QAnywhere addresses these challenges with mobile-optimized asynchronous web services that leverage the QAnywhere store-and-forward messaging architecture. By using QAnywhere mobile web services, your mobile applications can make web service requests, even when they are offline, and have those requests queued up for transmission later. The requests are delivered as QAnywhere messages and then a web services connector on the server side makes the request, gets the response from the web service, and returns the response to the client as a message. QAnywhere transmission rules can control which requests and responses are transmitted based on a wide variety of parameters (network being used, size of request/response, location, time of day, and so on). The result is a sophisticated and flexible architecture that allows mobile applications to tap into the vast functionality of web services using proven technology and a simple programming model.

From a development point of view, you can work with web service proxy classes much as you would in a connected environment and QAnywhere handles the transmission, authentication, serialization, and so on. A WSDL compiler is provided to take a WSDL document and generate special proxy classes (either .NET or Java) that a mobile application can use to invoke a web service. These classes use the underlying QAnywhere infrastructure to send requests and receive responses. When an object method call is made, a SOAP request is built automatically and delivered as a message to the server where a connector makes the web service request and returns the result as a message.

See also

- "Mobile web services" [*SQL Anywhere 12 - Introduction*]

Setting up mobile web services

Overview of setting up mobile web services

1. Set up a server message store, if you don't already have one.

   See "Server message store setup" on page 22.

2. Start the MobiLink server with the -m option and a connection to the server message store.

   See "Starting QAnywhere with MobiLink enabled" on page 29.

3. Set up client message stores, if you don't already have them. These are SQL Anywhere databases that are used to temporarily store messages.

   See "Setting up the client message store" on page 23.

4. Run the iAnywhere WSDL compiler to create classes you can use in your application.

   See "iAnywhere WSDL compiler" on page 90.

5. For each client, write a web service client application that uses the classes generated by the WSDL compiler.

   See "Mobile web service applications" on page 92.

6. Create a web services connector.

   See "Web service connectors" on page 137.

7. For each client, start the QAnywhere Agent (qaagent) with a connection to the local client message store.

   See "Starting the QAnywhere agent" on page 44.

Other resources for getting started

- An example showing how to set up a weather web service in Java is described in "Mobile web service example".
- A mobile web service sample application using a currency exchange web service is installed to *%SQLANYSAMP12%\QAnywhere\MobileWebServices*. This sample is provided in both Java and C#.
- You can post questions on the SQL Anywhere Forum: http://sqlanywhere-forum.sybase.com

Mobile web services development tips

- For mobile web services .NET applications, proxy classes generated by the WSDL compiler must be compiled into the application executable. They cannot be compiled into their own assembly.
- For mobile web services Java applications, JDK 1.5.x must be used.

iAnywhere WSDL compiler

Given a WSDL source that describes a web service, the iAnywhere WSDL compiler generates a set of Java proxy classes, C# proxy classes, or SQL SOAP client procedures for SQL Anywhere that you include in your application.

The Java or C# classes generated by the WSDL compiler are intended for use with QAnywhere. These classes expose web service operations as method calls. The classes that are generated are:

- The main service binding class (this class inherits from ianywhere.qanywhere.ws.WSBase in the mobile web services runtime).
- A proxy class for each complex type specified in the WSDL file.

For information about the generated proxy classes, see:

- "QAnywhere .NET API reference for web services" on page 307
- "QAnywhere Java API reference for web services" on page 586

The WSDL compiler supports WSDL 1.1 and SOAP 1.1 over HTTP and HTTPS.

Syntax

wsdlc [*options*] *wsdl-uri*

wsdl-uri

This is the specification for the WSDL (Web Services Description Language) source (a URL or file).

Options

- **-h** Display help text.
- **-v** Display verbose information.
- **-o *output-directory*** Specify an output directory for generated files.
- **-l *language*** Specify a language for the generated files. This is one of **java**, **cs**, or **sql**. These options must be specified in lowercase letters.
- **-d** Display debug information that may be helpful when contacting iAnywhere customer support.

Java-specific options

- **-p *package*** Specify a package name. This permits you to override the default package name.

C#-specific options

- **-n *namespace*** Specify a namespace. This permits you to wrap the generated classes in a namespace of your choosing.

SQL-specific options

- **-f *filename*** (Required) Specify the name of the output SQL file to which the SQL statements are written. This operation overwrites any existing file of the same name.
- **-p=*prefix*** Specify a prefix for the generated function or procedure names. The default prefix is the service name followed by a period (for example, "WSDish.").
- **-x** Generate procedure definitions rather than function definitions.

Mobile web service applications

Your application sends a web service request to QAnywhere, which sends the request to the mobile web service connector in the MobiLink server. The connector sends the request to the web service or queues the request until the web service is available. When QAnywhere receives the response, it notifies your application or queues the response until your application is available.

.NET mobile web service application setup

Before using .NET with QAnywhere, you must make the following changes to your Visual Studio project:

- Add references to the QAnywhere .NET DLL and the mobile web services .NET DLL. This tells Visual Studio which DLL to include to find the code for the QAnywhere .NET API and the mobile web services .NET API.
- Add lines to your source code to reference the QAnywhere .NET API classes and the mobile web services .NET API classes. To use the QAnywhere .NET API, you must add a line to your source code to reference the data provider. You must add a different line for C# than for Visual Basic.

Add references to the QAnywhere .NET API and mobile web services API in a Visual Studio project

1. Start Visual Studio and open your project.
2. In the Solution Explorer window, right-click the **References** folder and click **Add Reference**.
3. On the **Browse** tab, locate *iAnywhere.QAnywhere.Client.dll* and *iAnywhere.QAnywhere.WS.dll* in the following directories:
   - .NET Framework 2.0: *%SQLANY12%\Assembly\V2*
   - .NET Compact Framework 2.0: *%SQLANY12%\ce\Assembly\V2*

   From the appropriate directory for your environment, select each DLL and click **Open**.
4. To verify that the DLLs are added to your project, expand the **References** tree in the Solution Explorer. *iAnywhere.QAnywhere.Client.dll* and *iAnywhere.QAnywhere.WS.dll* should appear in the list.

Referencing the data provider classes in your source code

Reference the QAnywhere .NET API and mobile web services API classes in your code

1. Start Visual Studio and open your project.
2. If you are using C#, add the following lines to the list of using directives at the beginning of your file:

```
using iAnywhere.QAnywhere.Client;
using iAnywhere.QAnywhere.WS;
```


3. If you are using Visual Basic, add the following lines to the list of imports at the beginning of your file:

```
Imports iAnywhere.QAnywhere.Client
Imports iAnywhere.QAnywhere.WS
```

The Imports lines are not strictly required. However, they allow you to use short forms for the QAnywhere and mobile web services classes. Without them, you can still use the fully qualified class name in your code. For example, the following code uses the long form:

```
iAnywhere.QAnywhere.Client.QAManager
mgr =
iAnywhere.QAnywhere.Client.QAManagerFactory.Instance.CreateQAManager(
"qa_manager.props" );
```

The following code uses the short forms:

```
QAManager mgr = QAManagerFactory.Instance.CreateQAManager(
 "qa_manager.props" );
```

Initialize QAnywhere and mobile web services for .NET

1. Include the *iAnywhere.QAnywhere.Client* and *iAnywhere.QAnywhere.WS* namespaces, as described in the previous procedure.

```
using iAnywhere.QAnywhere.Client;
using iAnywhere.QAnywhere.WS;
```

2. Create a QAManager object.

For example, to create a default QAManager object, invoke CreateQAManager:

```
QAManager mgr;
mgr = QAManagerFactory.Instance.CreateQAManager();
```

Tip
For maximum concurrency benefits, multi-threaded applications should create a QAManager for each thread. See "Multi-threading considerations" on page 80.

For more information about QAManagerFactory, see "QAManagerFactory class [QAnywhere .NET]" on page 266.

Alternatively, you can create a QAManager object that is customized using a properties file. The properties file is specified in the CreateQAManager method:

```
mgr = QAManagerFactory.Instance.CreateQAManager(
  "qa_mgr.props" );
```

where *qa_mgr.props* is the name of the properties file that resides on the remote device.

3. Initialize the QAManager object. For example:

```
mgr.Open(
   AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT);
```

The argument to the open method is an acknowledgement mode, which indicates how messages are to be acknowledged. It must be one of IMPLICIT_ACKNOWLEDGEMENT or EXPLICIT_ACKNOWLEDGEMENT.

QAnywhere messages used by mobile web services are not accessible to the mobile web services application. When using a QAManager in EXPLICIT_ACKNOWLEDGEMENT mode, use the Acknowledge method of WSResult to acknowledge the QAnywhere message that contains the result of a web services request. This method indicates that the application has successfully processed the response.

For more information about acknowledgement modes, see:

- "WSBase.SetQAManager method [QAnywhere .NET]" on page 312
- "WSResult.Acknowledge method [QAnywhere .NET]" on page 325

Note
Instead of creating a QAManager, you can create a QATransactionalManager. See "Implementing transactional messaging for .NET clients" on page 63.

4. Create an instance of the service binding class.

   The mobile web services WSDL compiler generates the service binding class from the WSDL document that defines the web service.

   The QAManager is used by the instance of the web service binding class to perform messaging operations in the process of making web service requests. You specify the connector address to use to send web service requests through QAnywhere by setting the property WS_CONNECTOR_ADDRESS of the service binding class. You configure each QAnywhere web service connector with the URL of a web service to connect to, and if an application needs web services located at more than one URL, configure the connector for each URL.

   For example:

   ```
   CurrencyConverterSoap service = new CurrencyConverterSoap( )
   service.SetQAManager(mgr);
   service.setProperty(
      "WS_CONNECTOR_ADDRESS",
      "ianywhere.connector.currencyconvertor\\");
   ```

   Note that the final \\ in the address must be included.

See also

- "QAnywhere .NET API reference for web services" on page 307
- "QAnywhere .NET API reference for clients" on page 181

Example

To initialize mobile web services, you must create a QAManager and create an instance of the service binding class. For example:

```
// QAnywhere initialization
  QAManager mgr = QAManagerFactory.Instance.CreateQAManager();
```



```
  mgr.SetProperty( "CONNECT_PARAMS",
"server=qanywhere;dbf=qanywhere.db;uid=ml_qa_user;pwd=qanywhere" );
  mgr.Open( AcknowledgementMode.IMPLICIT_ACKNOWLEDGEMENT );
  mgr.Start();

  // Instantiate the web service proxy
  CurrencyConvertorSoap service = new CurrencyConvertorSoap();
  service.SetQAManager( mgr );
  service.SetProperty( "WS_CONNECTOR_ADDRESS",
"ianywhere.connector.currencyconvertor\\" );
```

The response time for the CurrencyConvertor sample depends on the availability of the web service. Asynchronous web service requests are useful when the mobile web service application is not always available. With this method, a web service request is made by calling a method on the service binding class to place the request in an outgoing queue. The method returns a WSResult, which can be used to query the status of the response at a later time, even after the application has been restarted. See "Asynchronous web service requests" on page 99.

Setting up Java mobile web service applications

To create mobile web service applications in Java, you must complete the following initialization tasks.

Initialize QAnywhere and mobile web services for Java

1. Add the location of the following files to your classpath. By default, they are located in *%SQLANY12%\Java*:
   - *qaclient.jar*
   - *iawsrt.jar*
   - *jaxrpc.jar*
2. Import the ianywhere.qanywhere.client and ianywhere.qanywhere.ws packages:

   ```
   import ianywhere.qanywhere.client.*;
   import ianywhere.qanywhere.ws.*;
   ```
3. Create a QAManager object.

   ```
   QAManager mgr;
   mgr = QAManagerFactory.getInstance().createQAManager(null);
   ```

   You can also customize a QAManager object by specifying a properties file to the createQAManager method:

   ```
   mgr = QAManagerFactory.getInstance().createQAManager("qa_mgr.props.");
   ```

   Tip
   For maximum concurrency benefits, multi-threaded applications should create a QAManager for each thread. See "Multi-threading considerations" on page 80.
4. Initialize the QAManager object.

   ```
   mgr.open(AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT);
   ```

The argument to the open method is an acknowledgement mode, which indicates how messages are to be acknowledged. It must be one of IMPLICIT_ACKNOWLEDGEMENT or EXPLICIT_ACKNOWLEDGEMENT.

QAnywhere messages used by mobile web services are not accessible to the mobile web services application. When using a QAManager in EXPLICIT_ACKNOWLEDGEMENT mode, use the Acknowledge method of WSResult to acknowledge the QAnywhere message that contains the result of a web services request. This method indicates that the application has successfully processed the response.

For more information about acknowledgement modes, see:

- "WSBase.setQAManager method [QAnywhere Java]" on page 590
- "WSResult.acknowledge method [QAnywhere Java]" on page 599

Note
Instead of creating a QAManager, you can create a QATransactionalManager. See "Implementing transactional messaging for Java clients" on page 66.

5. Create an instance of the service binding class.

   The mobile web services WSDL compiler generates the service binding class from the WSDL document that defines the web service.

   In the process of making web service requests, the QAManager is used by the instance of the web service binding class to perform messaging operations. You specify the connector address to use to send web service requests through QAnywhere by setting the WS_CONNECTOR_ADDRESS property of the service binding class. Each QAnywhere web service connector is configured with a URL of a web service to connect to. This means that if an application needs web services located at more than one URL, then a QAnywhere connector must be configured for each service URL.

   For example:

   ```
   CurrencyConverterSoap service = new CurrencyConverterSoap( );
   service.setQAManager(mgr);
   service.setProperty("WS_CONNECTOR_ADDRESS",
   "ianywhere.connector.currencyconvertor\\");
   ```

   Note that the final \\ in the address must be included.

See also

- "QAnywhere Java API reference for web services" on page 586
- "QAnywhere Java API reference for clients" on page 468

Example

To initialize mobile web services, you must create a QAManager and create an instance of the service binding class. For example:

```
// QAnywhere initialization
  Properties props = new Properties();
  props.put( "CONNECT_PARAMS",
```



```
"server=qanywhere;dbf=qanywhere.db;uid=ml_qa_user;pwd=qanywhere" );
  QAManager mgr = QAManagerFactory.getInstance().createQAManager( props );
  mgr.open( AcknowledgementMode.IMPLICIT_ACKNOWLEDGEMENT );
  mgr.start();

  // Instantiate the web service proxy
  CurrencyConvertorSoap service = new CurrencyConvertorSoap();
  service.setQAManager( mgr );
  service.setProperty( "WS_CONNECTOR_ADDRESS",
"ianywhere.connector.currencyconvertor\\" );
```

Multiple instances of the service binding class

You should create an instance of the service binding class for each QAManager. If a mobile web services application has more than one instance of a service binding class, it is important that the service ID be set using the SetServiceID method. For example:

```
service1.SetServiceID("1")
service2.SetServiceID("2")
```

The service ID is combined with the service name to form a queue name for receiving web service responses. It is important that each instance of a given service has a unique service ID so that a given instance does not get responses to requests made by another instance of the service. If the service ID is not set, it defaults to "". The service ID is also important for preventing multiple applications that use the same service from conflicting with each other, since queue names persist messages in the message store across applications that are transient.

Compiling and running mobile web service applications

Runtime libraries

The runtime library for Java is *iawsrt.jar*, located in *%SQLANY12%\Java*.

The runtime library for C# is *iAnywhere.QAnywhere.WS.dll*, located in the following directories:

- .NET Framework 2.0: *%SQLANY12%\Assembly\V2*
- .NET Compact Framework 2.0: *%SQLANY12%\ce\Assembly\V2*

The following sections describe the files you need to compile and run mobile web service applications.

Required runtime libraries (Java)

Include the following files in your classpath. They are located in *%SQLANY12%\Java*:

- *jaxrpc.jar*
- *qaclient.jar*
- *iawsrt.jar*

Required runtime libraries (.NET)

The SQL Anywhere 12 installation automatically includes the following files in your Global Assembly Cache:

- *iAnywhere.QAnywhere.Client.dll*
- *iAnywhere.QAnywhere.WS.dll*

Shutting down mobile web services

A mobile web services application performs orderly shutdown by closing the QAManager. For example:

```
// QAnywhere finalization in C#:
mgr.Stop();
mgr.Close();

// QAnywhere finalization in Java:
mgr.stop();
mgr.close();
```

Web service requests

There are two basic methods of making web service requests in a mobile web services application:

- **Synchronous** See “Synchronous web service requests”.
- **Asynchronous** See “Asynchronous web service requests”.

Synchronous web service requests

Synchronous web service requests are used when the application is connected to a network. With this method, a web service request is made by calling a method on the service binding class, and the result is returned only when the web service response has been received from the server.

Example

The following example makes a request to get the USD-to-CAD exchange rate:

```
//C#
double r = service.ConversionRate( Currency.USD, Currency.CAD );

// Java
double r = service.conversionRate( NET.webserviceX.Currency.USD,
NET.webserviceX.Currency.CAD );
```


Asynchronous web service requests

Asynchronous web service requests are useful when the mobile web service application is only occasionally connected to a network. With this method, a web service request is made by calling a method on the service binding class to place the request in an outgoing queue. The method returns a WSResult, which can be used to query the status of the response at a later time, even after the application has been restarted.

The following example makes an asynchronous request to get the USD-to-CAD exchange rate:

```
// C#
WSResult r = service.AsyncConversionRate( Currency.USD, Currency.CAD );

// Get the request ID.  Save it for later use if necessary.
string reqID = r.GetRequestID();

// Later: get the response for the specified request ID
WSResult r = service.GetResult( reqID );
if( r.GetStatus() == WSStatus.STATUS_RESULT_AVAILABLE ) {
  Console.WriteLine( "The conversion rate is " +
r.GetDoubleValue( "ConversionRateResult" ) );
} else {
  Console.WriteLine( "Response not available" );
}
// Java
WSResult r = service.asyncConversionRate( NET.webserviceX.Currency.USD,
NET.webserviceX.Currency.CAD );

// Get the request ID.  Save it for later use if necessary.
String reqID = r.getRequestID();

// Later: get the response for the specified request ID
WSResult r = service.getResult( reqID );
if( r.getStatus() == WSStatus.STATUS_RESULT_AVAILABLE ) {
    System.out.println( "The conversion rate is " +
r.getDoubleValue( "ConversionRateResult" ) );
} else {
    System.out.println( "Response not available" );
}
```

It is also possible to use a WSListener to get an asynchronous callback when the response to a web service request is available. For example:

```
// C#
// Make a request to get the USD to CAD exchange rate
WSResult r = service.AsyncConversionRate( Currency.USD, Currency.CAD );

// Register a listener for the result
service.SetListener( r.GetRequestID(), new CurrencyConvertorListener() );

// Java
// Make a request to get the USD to CAD exchange rate
WSResult r = service.asyncConversionRate( NET.webserviceX.Currency.USD,
NET.webserviceX.Currency.CAD );

// Register a listener for the result
service.setListener( r.getRequestID(), new CurrencyConvertorListener() );
```

The WSListener interface defines two methods for handling asynchronous events:

- **OnResult** An OnResult method is implemented to handle a response to a web service request. It is passed a WSResult object that represents the result of the web service request.

- **OnException** An OnException method is implemented to handle errors that occurred during processing of the response to the web service request. It is passed a WSException object and a WSResult object. The WSException object contains information about the error that occurred, and the WSResult object can be used to obtain the request ID that the response corresponds to.

```
// C#
class CurrencyConvertorListener : WSListener
{
   public CurrencyConvertorListener() {
   }

   public void OnResult( WSResult r ) {
    try {
         USDToCAD._statusMessage = "USD to CAD currency exchange rate: " +
r.GetDoubleValue( "ConversionRateResult" );
    } catch( Exception exc ) {
        USDToCAD._statusMessage = "Request " + r.GetRequestID() + " failed:
" + exc.Message;
    }
   }

   public void OnException( WSException exc, WSResult r ) {
    USDToCAD._statusMessage = "Request " + r.GetRequestID() + " failed: " +
exc.Message;
   }
}
// Java
private class CurrencyConvertorListener implements WSListener
{
  public CurrencyConvertorListener() {
  }

  public void onResult( WSResult r ) {
   try {
         USDToCAD._statusMessage = "USD to CAD currency exchange rate: " +
r.getDoubleValue( "ConversionRateResult" );
   } catch( Exception exc ) {
         USDToCAD._statusMessage = "Request " + r.getRequestID() + " failed:
" + exc.getMessage();
   }
   }

   public void onException( WSException exc, WSResult r ) {
         USDToCAD._statusMessage = "Request " + r.getRequestID() + " failed:
" + exc.getMessage();
   }
}
```

Mobile web service example

This example shows you how to create a mobile web service application. The application, which takes just a few minutes to create, uses the QAnywhere store-and-forward functionality so that you can issue a request for a weather report even if you are offline, and then see the report when it is available.

Global Weather web service

The following code describes a web service called Global Weather. (It is a wsdl file that was copied from a public weather web service.) Copy the code into a file and name the file *globalweather.wsdl*:

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:s="http://
www.w3.org/2001/XMLSchema" xmlns:soapenc="http://schemas.xmlsoap.org/soap/
encoding/" xmlns:tns="http://www.webserviceX.NET" xmlns:tm="http://
microsoft.com/wsdl/mime/textMatching/" xmlns:mime="http://
schemas.xmlsoap.org/wsdl/mime/" targetNamespace="http://www.webserviceX.NET"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
    <s:schema elementFormDefault="qualified" targetNamespace="http://
www.webserviceX.NET">
      <s:element name="GetWeather">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="CityName"
type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="CountryName"
type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="GetWeatherResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="GetWeatherResult"
type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="GetCitiesByCountry">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="CountryName"
type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="GetCitiesByCountryResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1"
name="GetCitiesByCountryResult" type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="string" nillable="true" type="s:string" />
    </s:schema>
  </wsdl:types>
  <wsdl:message name="GetWeatherSoapIn">
    <wsdl:part name="parameters" element="tns:GetWeather" />
  </wsdl:message>
  <wsdl:message name="GetWeatherSoapOut">
    <wsdl:part name="parameters" element="tns:GetWeatherResponse" />
  </wsdl:message>
  <wsdl:message name="GetCitiesByCountrySoapIn">
    <wsdl:part name="parameters" element="tns:GetCitiesByCountry" />
  </wsdl:message>
  <wsdl:message name="GetCitiesByCountrySoapOut">
```

```
      <wsdl:part name="parameters" element="tns:GetCitiesByCountryResponse" />
    </wsdl:message>
    <wsdl:message name="GetWeatherHttpGetIn">
      <wsdl:part name="CityName" type="s:string" />
      <wsdl:part name="CountryName" type="s:string" />
    </wsdl:message>
    <wsdl:message name="GetWeatherHttpGetOut">
      <wsdl:part name="Body" element="tns:string" />
    </wsdl:message>
    <wsdl:message name="GetCitiesByCountryHttpGetIn">
      <wsdl:part name="CountryName" type="s:string" />
    </wsdl:message>
    <wsdl:message name="GetCitiesByCountryHttpGetOut">
      <wsdl:part name="Body" element="tns:string" />
    </wsdl:message>
    <wsdl:message name="GetWeatherHttpPostIn">
      <wsdl:part name="CityName" type="s:string" />
      <wsdl:part name="CountryName" type="s:string" />
    </wsdl:message>
    <wsdl:message name="GetWeatherHttpPostOut">
      <wsdl:part name="Body" element="tns:string" />
    </wsdl:message>
    <wsdl:message name="GetCitiesByCountryHttpPostIn">
      <wsdl:part name="CountryName" type="s:string" />
    </wsdl:message>
    <wsdl:message name="GetCitiesByCountryHttpPostOut">
      <wsdl:part name="Body" element="tns:string" />
    </wsdl:message>
    <wsdl:portType name="GlobalWeatherSoap">
      <wsdl:operation name="GetWeather">
        <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Get weather
report for all major cities around the world.</documentation>
        <wsdl:input message="tns:GetWeatherSoapIn" />
        <wsdl:output message="tns:GetWeatherSoapOut" />
      </wsdl:operation>
      <wsdl:operation name="GetCitiesByCountry">
        <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Get all major
cities by country name(full / part).</documentation>
        <wsdl:input message="tns:GetCitiesByCountrySoapIn" />
        <wsdl:output message="tns:GetCitiesByCountrySoapOut" />
      </wsdl:operation>
    </wsdl:portType>
    <wsdl:portType name="GlobalWeatherHttpGet">
      <wsdl:operation name="GetWeather">
        <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Get weather
report for all major cities around the world.</documentation>
        <wsdl:input message="tns:GetWeatherHttpGetIn" />
        <wsdl:output message="tns:GetWeatherHttpGetOut" />
      </wsdl:operation>
      <wsdl:operation name="GetCitiesByCountry">
        <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Get all major
cities by country name(full / part).</documentation>
        <wsdl:input message="tns:GetCitiesByCountryHttpGetIn" />
        <wsdl:output message="tns:GetCitiesByCountryHttpGetOut" />
      </wsdl:operation>
    </wsdl:portType>
    <wsdl:portType name="GlobalWeatherHttpPost">
      <wsdl:operation name="GetWeather">
        <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Get weather
report for all major cities around the world.</documentation>
        <wsdl:input message="tns:GetWeatherHttpPostIn" />
        <wsdl:output message="tns:GetWeatherHttpPostOut" />
      </wsdl:operation>
      <wsdl:operation name="GetCitiesByCountry">
```



```
      <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Get all major
cities by country name(full / part).</documentation>
      <wsdl:input message="tns:GetCitiesByCountryHttpPostIn" />
      <wsdl:output message="tns:GetCitiesByCountryHttpPostOut" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="GlobalWeatherSoap" type="tns:GlobalWeatherSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="document" />
    <wsdl:operation name="GetWeather">
      <soap:operation soapAction="http://www.webserviceX.NET/GetWeather"
style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="GetCitiesByCountry">
      <soap:operation soapAction="http://www.webserviceX.NET/
GetCitiesByCountry" style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="GlobalWeatherHttpGet" type="tns:GlobalWeatherHttpGet">
    <http:binding verb="GET" />
    <wsdl:operation name="GetWeather">
      <http:operation location="/GetWeather" />
      <wsdl:input>
        <http:urlEncoded />
      </wsdl:input>
      <wsdl:output>
        <mime:mimeXml part="Body" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="GetCitiesByCountry">
      <http:operation location="/GetCitiesByCountry" />
      <wsdl:input>
        <http:urlEncoded />
      </wsdl:input>
      <wsdl:output>
        <mime:mimeXml part="Body" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="GlobalWeatherHttpPost"
type="tns:GlobalWeatherHttpPost">
    <http:binding verb="POST" />
    <wsdl:operation name="GetWeather">
      <http:operation location="/GetWeather" />
      <wsdl:input>
        <mime:content type="application/x-www-form-urlencoded" />
      </wsdl:input>
      <wsdl:output>
        <mime:mimeXml part="Body" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="GetCitiesByCountry">
```

```
      <http:operation location="/GetCitiesByCountry" />
      <wsdl:input>
        <mime:content type="application/x-www-form-urlencoded" />
      </wsdl:input>
      <wsdl:output>
        <mime:mimeXml part="Body" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="GlobalWeather">
    <wsdl:port name="GlobalWeatherSoap" binding="tns:GlobalWeatherSoap">
      <soap:address location="http://www.webservicex.net/
globalweather.asmx" />
    </wsdl:port>
    <wsdl:port name="GlobalWeatherHttpGet"
binding="tns:GlobalWeatherHttpGet">
      <http:address location="http://www.webservicex.net/
globalweather.asmx" />
    </wsdl:port>
    <wsdl:port name="GlobalWeatherHttpPost"
binding="tns:GlobalWeatherHttpPost">
      <http:address location="http://www.webservicex.net/
globalweather.asmx" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Generate proxy class

To create a mobile application to access the Global Weather web service, first run the QAnywhere WSDL compiler. It generates a proxy class that can be used in an application to make requests of the global weather service. In this example, the application is written in Java.

```
wsdlc -l java globalweather.wsdl
```

This command generates a proxy class called *GlobalWeatherSoap.java*, located in the *NET\webserviceX* subdirectory of the current directory. This proxy class is the service binding class for your application. The following is the content of *GlobalWeatherSoap.java*:

```
/*
 * GlobalWeatherSoap.java
 *
 * Generated by the iAnywhere WSDL Compiler Version 10.0.1.3415
 * Do not edit this file.
 */

package NET.webserviceX;

import ianywhere.qanywhere.ws.*;
import ianywhere.qanywhere.client.QABinaryMessage;
import ianywhere.qanywhere.client.QAException;

import java.io.*;

import javax.xml.transform.*;
import javax.xml.transform.sax.*;
import javax.xml.transform.stream.*;

public class GlobalWeatherSoap extends ianywhere.qanywhere.ws.WSBase
{

    public GlobalWeatherSoap(String iniFile) throws WSException
```



```
    {
        super(iniFile);
        init();
    }

    public GlobalWeatherSoap() throws WSException
    {
        init();
    }

    public void init()
    {

        setServiceName("GlobalWeather");
    }


    public java.lang.String getWeather(java.lang.String cityName,
                        java.lang.String countryName) throws QAException,
WSException, WSFaultException
    {
        try {
            StringWriter sw = new StringWriter();
            SAXTransformerFactory stf = (SAXTransformerFactory)
SAXTransformerFactory.newInstance();
            TransformerHandler hd = stf.newTransformerHandler();
            QABinaryMessage qaRequestMsg = null;

            hd.setResult( new StreamResult( sw ) );
            String        responsePartName = "GetWeatherResult";
            java.lang.String    returnValue;

            writeSOAPHeader( hd, "GetWeather", "http://
www.webserviceX.NET" );

WSBaseTypeSerializer.serialize(hd,"CityName",cityName,"string","http://
www.w3.org/2001/XMLSchema",true,true);

WSBaseTypeSerializer.serialize(hd,"CountryName",countryName,"string","http://
www.w3.org/2001/XMLSchema",true,true);
            writeSOAPFooter( hd, "GetWeather" );

            qaRequestMsg = createQAMessage( sw.toString(), "http://
www.webserviceX.NET/GetWeather", "GetWeatherResponse" );

            WSResult wsResult = invokeWait( qaRequestMsg );

            returnValue = wsResult.getStringValue(responsePartName);

            return returnValue;
        } catch( TransformerConfigurationException e ) {
            throw new WSException( e );
        }
    }

    public WSResult asyncGetWeather(java.lang.String cityName,
                        java.lang.String countryName) throws QAException,
WSException
    {
        try {
            StringWriter sw = new StringWriter();
            SAXTransformerFactory stf = (SAXTransformerFactory)
SAXTransformerFactory.newInstance();
            TransformerHandler hd = stf.newTransformerHandler();
```

```
            QABinaryMessage qaRequestMsg = null;

            hd.setResult( new StreamResult( sw ) );

            writeSOAPHeader( hd, "GetWeather", "http://
www.webserviceX.NET" );

WSBaseTypeSerializer.serialize(hd,"CityName",cityName,"string","http://
www.w3.org/2001/XMLSchema",true,true);

WSBaseTypeSerializer.serialize(hd,"CountryName",countryName,"string","http://
www.w3.org/2001/XMLSchema",true,true);
            writeSOAPFooter( hd, "GetWeather" );

            qaRequestMsg = createQAMessage( sw.toString(), "http://
www.webserviceX.NET/GetWeather", "GetWeatherResponse" );

            WSResult wsResult = invoke( qaRequestMsg );

            return wsResult;
        } catch( TransformerConfigurationException e ) {
            throw new WSException( e );
        }
    }


    public java.lang.String getCitiesByCountry(java.lang.String countryName)
throws QAException, WSException, WSFaultException
    {
        try {
            StringWriter sw = new StringWriter();
            SAXTransformerFactory stf = (SAXTransformerFactory)
SAXTransformerFactory.newInstance();
            TransformerHandler hd = stf.newTransformerHandler();
            QABinaryMessage qaRequestMsg = null;

            hd.setResult( new StreamResult( sw ) );
            String        responsePartName = "GetCitiesByCountryResult";
            java.lang.String    returnValue;

            writeSOAPHeader( hd, "GetCitiesByCountry", "http://
www.webserviceX.NET" );

WSBaseTypeSerializer.serialize(hd,"CountryName",countryName,"string","http://
www.w3.org/2001/XMLSchema",true,true);
            writeSOAPFooter( hd, "GetCitiesByCountry" );

            qaRequestMsg = createQAMessage( sw.toString(), "http://
www.webserviceX.NET/GetCitiesByCountry", "GetCitiesByCountryResponse" );

            WSResult wsResult = invokeWait( qaRequestMsg );

            returnValue = wsResult.getStringValue(responsePartName);

            return returnValue;
        } catch( TransformerConfigurationException e ) {
            throw new WSException( e );
        }
    }

    public WSResult asyncGetCitiesByCountry(java.lang.String countryName)
throws QAException, WSException
    {
        try {
```



```
            StringWriter sw = new StringWriter();
            SAXTransformerFactory stf = (SAXTransformerFactory)
SAXTransformerFactory.newInstance();
            TransformerHandler hd = stf.newTransformerHandler();
            QABinaryMessage qaRequestMsg = null;

            hd.setResult( new StreamResult( sw ) );

            writeSOAPHeader( hd, "GetCitiesByCountry", "http://
www.webserviceX.NET" );

WSBaseTypeSerializer.serialize(hd,"CountryName",countryName,"string","http://
www.w3.org/2001/XMLSchema",true,true);
            writeSOAPFooter( hd, "GetCitiesByCountry" );

            qaRequestMsg = createQAMessage( sw.toString(), "http://
www.webserviceX.NET/GetCitiesByCountry", "GetCitiesByCountryResponse" );

            WSResult wsResult = invoke( qaRequestMsg );

            return wsResult;
        } catch( TransformerConfigurationException e ) {
            throw new WSException( e );
        }
    }
}
```

Write mobile web service applications

Next, write applications that use the service binding class to make requests of the web service and process the results. Following are two applications, both of which make web service requests offline and process the results when a connection is available.

The first application, called RequestWeather, makes a request of the global weather service and displays the ID of the request. Copy the following code into a file called *RequestWeather.java*:

```
import ianywhere.qanywhere.client.*;
import ianywhere.qanywhere.ws.*;
import NET.webserviceX.GlobalWeatherSoap;

class RequestWeather
{
    public static void main( String [] args ) {
    try {
        // QAnywhere initialization
        QAManager mgr = QAManagerFactory.getInstance().createQAManager();
        mgr.open( AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT );
        mgr.start();

        // Instantiate the web service proxy
        GlobalWeatherSoap service = new GlobalWeatherSoap();
        service.setQAManager( mgr );
        service.setProperty( "WS_CONNECTOR_ADDRESS",
"ianywhere.connector.globalweather\\" );

        // Make a request to get weather for Beijing
        WSResult r = service.asyncGetWeather( "Beijing", "China" );

        // Display the request ID so that it can be used by ShowWeather
        System.out.println( "Request ID: " + r.getRequestID() );

        // QAnywhere finalization
```

```
            mgr.stop();
            mgr.close();

        } catch( Exception exc ) {
            System.out.println( exc.getMessage() );
        }
        }
}
```

The second application, called ShowWeather, shows the weather conditions for a given request ID. Copy the following code into a file called *ShowWeather.java*:

```
import ianywhere.qanywhere.client.*;
import ianywhere.qanywhere.ws.*;
import NET.webserviceX.GlobalWeatherSoap;

class ShowWeather
{
   public static void main( String [] args ) {
   try {
     // QAnywhere initialization
     QAManager mgr = QAManagerFactory.getInstance().createQAManager();
     mgr.open( AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT );
     mgr.start();

     // Instantiate the web service proxy
     GlobalWeatherSoap service = new GlobalWeatherSoap();
     service.setQAManager( mgr );

     // Get the response for the specified request ID
     WSResult r = service.getResult( args[0] );
     if( r.getStatus() == WSStatus.STATUS_RESULT_AVAILABLE ) {
  System.out.println( "The weather is " +
r.getStringValue( "GetWeatherResult" ) );
  r.acknowledge();
     } else {
  System.out.println( "Response not available" );
     }

     // QAnywhere finalization
     mgr.stop();
     mgr.close();

 } catch( Exception exc ) {
     System.out.println( exc.getMessage() );
 }
    }
}
```

Compile the application and the service binding class:

```
javac -classpath ".;%sqlany12%\java\iawsrt.jar;%sqlany12%\java\qaclient.jar"
NET.webserviceX.GlobalWeatherSoap.java RequestWeather.java
javac -classpath ".;%sqlany12%\java\iawsrt.jar;%sqlany12%\java\qaclient.jar"
NET.webserviceX.GlobalWeatherSoap.java ShowWeather.java
```

Create QAnywhere message stores and start a QAnywhere Agent

Your mobile web service application requires a client message store on each mobile device. It also requires a server message store, but this example uses the QAnywhere sample server message store.

To create a client message store, create a SQL Anywhere database with the dbinit utility and then run the QAnywhere Agent to set it up as a client message store:

```
dbinit -i qanywhere.db
qaagent -q -si -c "dbf=qanywhere.db"
```

Start the QAnywhere Agent to connect to your client message store:

```
qaagent -c "dbf=qanywhere.db;server=qanywhere;uid=ml_qa_user;pwd=qanywhere"
```

Start the QAnywhere server:

```
mlsrv12 -m -zu+ -c "dsn=QAnywhere 12
Demo;uid=ml_server;pwd=sql;start=dbsrv12" -v+ -ot qanyserv.mls
```

Create a web service connector

Create a web service connector that listens for QAnywhere messages sent to the GetWeather web service, makes web service calls when messages arrive, and sends back responses to the originating client.

1. Open Sybase Central and click **Connections** » **Connect With QAnywhere 12**.
2. In the **User ID** field, type **ml_server**.
3. In the **Password** field, type **sql**.
4. Click **ODBC data source name** and type **QAnywhere 12 Demo**.
5. Click **Connect**.
6. Click **File** » **New** » **Connector**.
7. Click **Web Services**. Click **Next**.
8. In the **Connector Name** field, type **ianywhere.connector.globalweather**. Click **Next**.
9. In the **URL** field, type **http://www.webservicex.net/globalweather.asmx**. Click **Finish**.

Use the web service

To queue up a request to get the weather report from the web service, type:

```
java -classpath ".;%sqlany12%\java\iawsrt.jar;%sqlany12%\java\qaclient.jar"
RequestWeather
```

A request ID is returned.

To see the weather report, type the following. The end should be the request ID, which in this example is REQ123123123.

```
java -classpath ".;%sqlany12%\java\iawsrt.jar;%sqlany12%\java\qaclient.jar"
ShowWeather REQ123123123
```

A detailed weather report is returned.

QAnywhere application deployment

QAnywhere provides C++, Java, and .NET API support for SQL Anywhere message stores. The Java and .NET APIs also support UltraLite message stores. The files required for deploying QAnywhere applications are based on your Windows environment, message store type, and API selection. Additional files are required if you are developing Mobile Web Service applications.

In addition to the files listed below, a QAnywhere application requires:

- All files listed in the MobiLink synchronization client, Listener, and optionally the Security sections of "SQL Anywhere MobiLink client deployment" [*MobiLink - Server Administration*].

  The MobiLink Listener files are required only if you are using push notifications, which is the default.

- dbeng12 or dbsrv12 files from "Database server deployment" [*SQL Anywhere Server - Programming*].

To deploy Sybase Central, see "Administration tool deployment" [*SQL Anywhere Server - Programming*].

Windows applications

All directories are relative to *%SQLANY12%*.

For more details on the file structure of a Windows Mobile environment, see "Windows Mobile applications" [*MobiLink - Server Administration*].

The following is a list of files required to set up a SQL Anywhere message store.

Client API	Windows files
C++	• *bin32\qany12.dll* • *bin32\qaagent.exe* • *bin32\qastop.exe*
Java	• *bin32\qaagent.exe* • *bin32\qastop.exe* • *java\qaclient.jar* • *java\jodbc.jar* For Mobile Web Service applications, you also need the following: • *java\iawsrt.jar* • *java\jaxrpc.jar*

Client API	Windows files
.NET	• *bin32\qazlib.dll* • *bin32\qaagent.exe* • *bin32\qastop.exe* • *assembly\v2\iAnywhere.QAnywhere.Client.dll* • *assembly\v2\iAnywhere.QAnywhere.Resources.dll* • *assembly\v2\iAnywhere.Data.SQLAnywhere.dll* For Mobile Web Service applications, you also need the following: • *Assembly\v2\iAnywhere.QAnywhere.WS.dll*

The following is a list of files required to set up an UltraLite message store.

Client API	Windows files
Java	• *bin32\qauagent.exe* • *bin32\qastop.exe* • *bin32\qadbiuljni12.dll* • *java\qaclient.jar* For Mobile Web Service applications, you also need the following: • *java\iawsrt.jar* • *java\jaxrpc.jar*
.NET	• *bin32\qazlib.dll* • *bin32\qauagent.exe* • *bin32\qastop.exe* • *assembly\v2\iAnywhere.QAnywhere.Client.dll* • *assembly\v2\iAnywhere.QAnywhere.Resources.dll* • *ultralite\ultralite.NET\assembly\v2\iAnywhere.Data.UltraLite.dll* For Mobile Web Service applications, you also need the following: • *Assembly\v2\iAnywhere.QAnywhere.WS.dll*

When creating an UltraLite message store, you must create a udb database file using the UltraLite Create Database utility, then initialize the database using the QAnywhere UltraLite Agent's -si option. See "UltraLite Initialize Database utility (ulinit)" [*UltraLite - Database Management and Reference*] and "qauagent utility" on page 696.

Windows Mobile applications

All directories are relative to *%SQLANY12%*.

For more details on the file structure of a Windows environment, see "Windows applications" [*MobiLink - Server Administration*].

The following is a list of files required to set up a SQL Anywhere message store.

Client API	Windows Mobile files
C++	• *ce\arm.50\qany12.dll* • *ce\arm.50\qaagent.exe* • *ce\arm.50\qastop.exe*
Java	• *ce\arm.50\qaagent.exe* • *ce\arm.50\qastop.exe* • *java\qaclient.jar* • *java\jodbc.jar* For Mobile Web Service applications, you also need the following: • *java\iawsrt.jar* • *java\jaxrpc.jar*
.NET	• *ce\arm.50\qazlib.dll* • *ce\arm.50\qaagent.exe* • *ce\arm.50\qastop.exe* • *ce\assembly\v2\iAnywhere.QAnywhere.Client.dll* • *ce\assembly\v2\iAnywhere.QAnywhere.Resources.dll* • *ce\assembly\v2\iAnywhere.Data.SQLAnywhere.dll* For Mobile Web Service applications, you also need the following: • *ce\Assembly\v2\iAnywhere.QAnywhere.WS.dll*

The following is a list of files required to set up an UltraLite message store.

Client API	Windows Mobile files
Java	• *ce\arm.50\qauagent.exe* • *ce\arm.50\qastop.exe* • *ce\arm.50\qadbiuljni12.dll* • *java\qaclient.jar* For Mobile Web Service applications, you also need the following: • *java\iawsrt.jar* • *java\jaxrpc.jar*

Client API	Windows Mobile files
.NET	• *ce\arm.50\qazlib.dll* • *ce\arm.50\qauagent.exe* • *ce\arm.50\qastop.exe* • *ce\assembly\v2\iAnywhere.QAnywhere.Client.dll* • *ce\assembly\v2\iAnywhere.QAnywhere.Resources.dll* • *ultralite\ultralite.NET\ce\assembly\v2\iAnywhere.Data.UltraLite.dll* For Mobile Web Service applications, you also need the following: • *ce\Assembly\v2\iAnywhere.QAnywhere.WS.dll*

When creating an UltraLite message store, you must create a database file using the UltraLite Create Database utility, then initialize the database using the -si option for the QAnywhere UltraLite Agent. See "UltraLite Initialize Database utility (ulinit)" [*UltraLite - Database Management and Reference*] and "qauagent utility" on page 696.

Registering the QAnywhere .NET API DLL

The QAnywhere .NET API DLL (*Assembly\v2\iAnywhere.QAnywhere.Client.dll*) needs to be registered in the Global Assembly Cache on Windows (except on Windows Mobile). The Global Assembly Cache lists all the registered programs on your computer. When you install SQL Anywhere, the installation program registers it. In Windows Mobile you do not need to register the DLL.

If you are deploying QAnywhere, you must register the QAnywhere .NET API DLL (*Assembly \v2\iAnywhere.QAnywhere.Client.dll*) using the gacutil utility that is included with the .NET Framework.

Secure messaging applications

The following sections describe ways of ensuring your messaging applications are secure.

Creating a secure client message store

To secure your client message store, you can:

- Change the default passwords.
- Encrypt the contents of the message store.

Example

First, create a SQL Anywhere database with an encryption key:

```
dbinit mystore.db -i -s -ek some_phrase
```

The -i and -s options are optimal for small devices. The -ek option specifies the encryption key for strong encryption. See "Initialization utility (dbinit)" [*SQL Anywhere Server - Database Administration*].

Next, initialize the database as a client message store:

```
qaagent -id mystore -si -c "dbf=mystore.db;dbkey=some_phrase"
```

Next, create a new remote user with DBA authority, and a password for this user. Revoke the default QAnywhere user and change the password of the default DBA user. Log in as user DBA with password sql and execute the following SQL statements:

```
CREATE USER secure_user IDENTIFIED BY secure_password
GRANT MEMBERSHIP IN GROUP ml_qa_user_group TO secure_user
GRANT REMOTE DBA TO secure_user
REVOKE CONNECT FROM ml_qa_user
ALTER USER DBA IDENTIFIED BY new_dba_password
COMMIT
```

Note
All QAnywhere users must belong to ml_qa_user_group and have remote DBA authority.

Next, start the QAnywhere Agent with the secure DBA user:

```
qaagent -id mystore -c
"dbf=mystore.db;dbkey=some_phrase;uid=secure_user;pwd=secure_password"
```

See also

- "Manage client message store passwords"
- "Client message store encryption"

Manage client message store passwords

You should change the passwords for the default user IDs that were created for the message store. The default user ID DBA with password SQL is created for every SQL Anywhere database. In addition, the qaagent -si option creates a default user ID of ml_qa_user, and creates a default password of qanywhere. To change these passwords, use the GRANT statement.

See "Changing a password" [*SQL Anywhere Server - Database Administration*].

Client message store encryption

The following command can be used to encrypt the client message store when you create it.

```
dbinit -i -s -ek encryption-key database-file
```

(The -i and -s options are good practice for creating databases on small devices.) When a message store has been initialized with an encryption key, the encryption key is required to start the database server on the encrypted message store.

Use the following command to specify the encryption key to start the QAnywhere Agent with an encrypted message store. The QAnywhere Agent automatically starts the database server on the encrypted message store using the encryption key provided.

```
qaagent -si -c "DBF=database-file;DBKEY=encryption-key"
```

Any application can now access the encrypted message store through the QAnywhere APIs. Note that, since the database server used to manage the message store is already running, the application does not need to provide the encryption key.

If the QAnywhere Agent is not running and an application needs to access an encrypted message store, the QAnywhere APIs automatically starts the database server using the connection parameters specified in the QAnywhere Manager initialization file. To start the database server on an encrypted message store, the encryption key must be specified in the database connection parameters as follows.

```
CONNECT_PARAMS=DBF=database-file;DBKEY=encryption-key
```

See also

- "Database encryption and decryption" [*SQL Anywhere Server - Database Administration*]
- "Initialization utility (dbinit)" [*SQL Anywhere Server - Database Administration*]
- QAnywhere Agent "-c qaagent option" on page 675

Communication stream encryption

The qaagent -x option can be used to specify a secure communication stream that the QAnywhere Agent can use to communicate with a MobiLink server. It allows you to implement server authentication using server-side certificates, and it allows you to encrypt the communication stream using strong encryption.

You must set up transport-layer security for the MobiLink server as well.

Note
Separately licensed component required.

ECC encryption and FIPS-certified encryption require a separate license. All strong encryption technologies are subject to export regulations.

See "Separately licensed components" [*SQL Anywhere 12 - Introduction*].

Examples

The following examples show how to establish a secure communication stream between the QAnywhere Agent and the MobiLink server. They use sample identity files that are installed when the SQL Anywhere security option is installed.

Secure TCP/IP using RSA:

```
mlsrv12 -x tls(tls_type=rsa;identity=rsaserver.id;identity_password=test)
qaagent -x tls(tls_type=rsa;trusted_certificate=rsaroot.crt)
```

Secure TCP/IP using ECC:

```
mlsrv12 -x tls(tls_type=ecc;identity=eccserver.id;identity_password=test)
qaagent -x tls(tls_type=ecc;trusted_certificate=eccroot.crt)
```

Secure HTTP using HTTPS (only RSA certificates are supported for HTTPS):

```
mlsrv12 -x https(identity=rsaserver.id;identity_password=test)
qaagent -x https(trusted_certificate=rsaroot.crt)
```

See also

- "-x qaagent option" on page 694
- "MobiLink client/server communications encryption" [*SQL Anywhere Server - Database Administration*]

Password authentication with MobiLink

Once you have established a secure communication stream between the remote device and the server, you may also want to authenticate the user of the device to ensure that they are allowed to communicate with the server.

You do this by creating a MobiLink user name for the client message store and registering it on the server message store.

See also

- "-mu qaagent option" on page 682
- "-mp qaagent option" on page 681
- "MobiLink users" [*MobiLink - Client Administration*]

Server management request security

Server management requests can be secured using a password. The message string property ias_ServerPassword specifies the server password. The server password is set using the ianywhere.qa.server.password.e property. If this property is not set, the password is QAnywhere.

The server password is transmitted as text. Use an encrypted communication stream to send server management requests that require a server password.

For more information about the ianywhere.qa.server.password.e property, see "Server properties" on page 724.

Add users with the MobiLink user authentication utility

To ensure security, use the MobiLink user authentication utility (mluser) to add users. The mluser utility allows you to register a MobiLink user name with the server message store on the consolidated database. The user name and password parameters are then used by the QAnywhere agent to authenticate the user during message transmission.

If you are using push notifications, it is also necessary to add a MobiLink user for the MobiLink Listener (dblsn). For each user added, a Listener must also be added with the username ias_[user]_lsn.

New users can also be added using the -zu+ option with mlsrv12, however you should not use this option if security is an issue. Using mlsrv12 with the -zu+ option causes all new users to be added to the consolidated database when they first synchronize. This means that unrecognized users are added without authentication.

See also

- "MobiLink User Authentication utility (mluser)" [*MobiLink - Server Administration*]
- "-zu mlsrv12 option" [*MobiLink - Server Administration*]

Security with the Relay Server

When using the Relay Server, set options in the Relay Server configuration file to control the level of security required.

Set up a secure communication stream with the web server, for example, HTTPS with trusted certificate, to ensure security of communications between the QAnywhere agent and the web server.

Refer to the Relay Server documentation for information about setting up a secure communication stream between the Relay Server and MobiLink.

See also

- "Relay Server configuration file" [*Relay Server*]
- "-x qaagent option" on page 694
- "Introduction to the Relay Server" [*Relay Server*]

Server message store administration

The following sections describe how to administer a server message store.

Transmission rules

Transmission rules allow you to specify when message transmission is to occur and which messages to transmit. You can specify transmission rules for both the client and the server.

Managing server transmission rules is an important part of administering a server message store.

Server transmission rules govern the behavior of messages going from the server to the client. Server transmission rules are handled by the MobiLink server. They apply both when you are using push notifications and when you are not using notifications.

There are several ways to set server transmission rules:

- Write a server management request to set the transmission rule.

  See "Transmission rule specification with a server management request" on page 162.

- Use Sybase Central to set the rules.

  See "Specifying server transmission rules using Sybase Central" on page 122.

- Create a server transmission rules file and specify it when you start the MobiLink server. This method is deprecated.

  See "Server transmission rule specification with a transmission rules file (deprecated)" on page 739.

Server transmission rules

Server transmission rules govern the behavior of messages going from the server to the client. Server transmission rules are handled by the MobiLink server. They apply both when you are using push notifications and when you are not using notifications.

There are several ways to set server transmission rules:

- Write a server management request to set the transmission rule.

  See "Transmission rule specification with a server management request" on page 162.

- Use Sybase Central to set the rules.

  See "Specifying server transmission rules using Sybase Central" on page 122.

- Create a server transmission rules file and specify it when you start the MobiLink server. This method is deprecated.

  See "Server transmission rule specification with a transmission rules file (deprecated)" on page 739.

Specifying server transmission rules using Sybase Central

Specify default server transmission rules

1. Start Sybase Central:
   - Click **Start** » **Programs** » **SQL Anywhere 12** » **Administration Tools** » **Sybase Central**.
   - From **Connections**, click **Connect With QAnywhere 12**.
   - Specify an **ODBC Data Source Name** or **ODBC Data Source File**, and the **User ID** and **Password** if required. Click **Connect**.
2. Under **Server Messages Stores**, click the data source name.
3. Click **File** » **Properties**.
4. Open the **Transmission Rules** tab and click **Customize The Default Transmission Rules**.
5. Click **New** to add a rule.
6. Add conditions either by typing them into the text field or by choosing **Message Variables** or **Constants** from the dropdown lists.
7. Click **OK** to exit.

Specifying server transmission rules with a server management request

You can use a server management request to specify default server transmission rules that apply to all users, or you can specify transmission rules for each client.

To specify default transmission rules for a server, set the ianywhere.qa.server.rules property for the client ianywhere.server.defaultClient. For a client, use the ianywhere.qa.server.rules property to specify server transmission rules.

See also

- “Transmission rule specification with a server management request” on page 162

Message archive management

The archive message store is a set of tables that coexist with the server message store, and stores all messages waiting to be deleted. A regularly executed system process transports messages between the message stores by removing all messages in the server message store that have reached a final state, and then inserting them into the archive message store.

Messages remain in the archive message store until deleted by a server delete rule. Usage of the archive message store improves the performance of the server message store by minimizing the amount of messages that need to be filtered during synchronization.

See also

- “Delete rules” on page 44

Server message store administration with server management requests

A server management request is a special message sent from the client to the server. The server management request contains content, formatted XML, that instructs the server to perform various functions.

Use the following functions to administer a server message store with server management requests:

- “Client transmission rule refresh” on page 150
- “Message cancellation” on page 151
- “Message deletion” on page 152

For information about using server management requests, see:

- “Server message store administration with server management requests” on page 150
- “Server management request writing” on page 148
- “Server management request reference” on page 664

Client message store administration

The following sections describe how to administer a client message store.

QAnywhere client monitoring

You can monitor QAnywhere clients using server management requests or Sybase Central.

Server management requests can be user to obtain a list of clients currently on the server. This list contains clients who are registered on the server, including remote clients, open connectors, and destination aliases.

In Sybase Central, use the **Clients** pane of the server message store to see a list of clients that are currently on the server.

See also

- "QAnywhere client monitoring" on page 168

Client properties monitoring

You can monitor QAnywhere client properties using server management requests or Sybase Central.

Server management requests can be used to see what properties are set for a client. The response lists only the properties that have been set for the client (not defaults).

In Sybase Central, you and view and change client properties using the QAnywhere **Client Properties** window.

See also

- "Property monitoring" on page 168

Client message store property management

Client message store properties can be set in your client application for each client message store.

Client message store properties can be used in transmission rules to filter messages to the client or used in delete rules to determine messages to add.

Client message store properties can also be specified in server management messages, and stored on the server message store.

See also

- “Management of client message store properties in your application” on page 722
- “QAnywhere transmission and delete rules” on page 729
- “Server management requests” on page 147

Destination aliases

A **destination alias** is a list of message addresses and other destination aliases. When a message is sent to a destination alias, it is sent to all members of the list.

A member of a destination alias can have a delivery condition associated with it. Only messages that match the condition are forwarded to the corresponding member.

Example

Define a destination alias called all_clients with members client1 and client2.

Define the following delivery condition for client1:

```
ias_Priority=1
```

Define the following delivery condition for client2:

```
ias_Priority=9
```

Only messages with priority 1 are sent to client1 and those with priority 9 are sent to client2.

Creating destination aliases

You can create and manage a destination alias using the following methods:

- Server management requests
- Sybase Central

Using Sybase Central

You can use Sybase Central to create or modify a destination alias.

Create a destination alias using Sybase Central

1. Start Sybase Central:
   - Click **Start** » **Programs** » **SQL Anywhere 12** » **Administration Tools** » **Sybase Central**.
   - Click **Connections** » **Connect with QAnywhere 12**.
   - Specify an **ODBC Data Source Name** or **ODBC Data Source File**, and a **User ID** and **Password** if required.
   - Click **OK**.
2. Click **File** » **New** » **Destination Alias**.
3. In the **Alias** field, type a name for the alias.
4. In the **Destinations** field, type the name of each destination on its own line.

5. Click **OK**.

See also

- "Destination alias creating with a server management request" on page 163
- "Using Sybase Central" on page 127

Connectors

The following sections provide information about using JMS connectors and web service connectors with QAnywhere.

JMS connectors

The Java Message Service (JMS) API provides messaging capabilities to Java applications. In addition to exchanging messages among QAnywhere client applications, you can exchange messages with external messaging systems that support a JMS interface. You do this using a specially configured client known as a connector. In a QAnywhere deployment, the external messaging system is set up to act like a QAnywhere client. It has its own address and configuration.

When running MobiLink with QAnywhere messaging in a server farm environment, only the QAnywhere connectors in the primary server start. If the primary server fails, the QAnywhere connectors are automatically started in the new primary server so that message ordering is preserved while exchanging data in the external messaging system, such as JMS.

See also

- "Scenario for messaging with external messaging systems" on page 7
- "Run the MobiLink server in a server farm" [*MobiLink - Server Administration*]

Setting up JMS connectors

The following steps provide an overview of the tasks required to set up QAnywhere with JMS connectors, assuming that you already have QAnywhere set up.

Overview of integrating a QAnywhere application with an external JMS system

1. Create JMS queues using the JMS administration tools for your JMS system. The QAnywhere connector listens on a single JMS queue for JMS messages. You must create this queue if it does not already exist.

   See the documentation of your JMS product for information about how to create queues.

2. Open Sybase Central.

3. Connect to your server message store using the QAnywhere plug-in.

   Click **Connections** » **Connect with QAnywhere 12**.

   The **Connect to a Generic ODBC Database** window appears

4. In the **User ID** field, type **ml_server**.

5. In the **Password** field, type **sql**.

6. Click **OK**.
7. Click **File** » **New** » **Connector**.
8. Click **JMS** and then click the type of web server you are using on the **Which JMS System Are You Using** list. Click **Next**.
9. On the **Connector Names** page:
   - In the **Connector Name** field, type the JMS System you are using. See "Sending a QAnywhere message to a JMS connector" on page 133.
   - In the **Receiver Destination** field, type **qanywhere_receive**, the queue name used by the connector to listen for messages from JMS targeted for QAnywhere clients.
   - Click **Next**.
10. On the **JNDI Settings** page:
    - In the **JNDI Factory** field, type the factory name used to access the external JMS JNDI name service.
    - In the **Name Service URL** field, type the URL to access the JMS JNDI name service.
    - In the **User Name** field, type the authentication name to connect to the external JMS JNDI name service.
    - In the **Password** field, type the authentication password to connect to the external JMS JNDI name service.
    - Click **Next**.
11. On the **JMS Queue Settings** page:
    - In the **Queue Factory** field, type the external JMS provider queue factory name.
    - In the **User Name** field, type the user ID to connect to the external JMS queue connection.
    - In the **Password** field, type the password to connect to the external JMS queue connection.
    - Click **Next**.
12. On the **JMS Topic Settings** page:
    - In the **Topic Factory** field, type the external JMS provider topic factory name.
    - In the **User Name** field, type the user ID to connect to the external JMS topic connection.
    - In the **Password** field, type the password to connect to the external JMS topic connection.
    - Click **Finish**.
13. Click **OK**.
14. Start the MobiLink server with a connection to the server message store and the -sl java options. See "Starting the MobiLink server for JMS integration" on page 131.

Starting the MobiLink server
When a JMS Connector is defined in this way, you must always start the MobiLink server as described, with the JMS client classes defined in the -sl java options. If the JMS Connector is no longer needed and you want to start MobiLink without the -sl java options, you must first remove the JMS Connector using Sybase Central.

15. To set additional options on your JMS connector, right-click the connector you just created and click properties; or you can use server management requests.

    For a list of available properties, see "JMS connector property configuration" on page 132.

    For information about how to set connector properties with server management requests, see "Connector administration with server management requests" on page 152.

Send messages

1. To send a message from an application in your QAnywhere system to the external messaging system, create a QAnywhere message and send it to ***connector-address\JMS-queue-name***.

   See "Sending a QAnywhere message to a JMS connector" on page 133.

2. To send a message from the external messaging system to an application in your QAnywhere system:

   - Create a JMS message.
   - Set the ias_ToAddress property to the QAnywhere ***id\queue*** (where *id* is the ID of your client message store and *queue* is your application queue name).
   - Put the message in the JMS queue.

   See "Addressing JMS messages meant for QAnywhere" on page 135.

Other resources for getting started

- QAnywhere JMS samples are installed to *%SQLANYSAMP12%\QAnywhere\jms*.
- You can post questions on the SQL Anywhere Forum: http://sqlanywhere-forum.sybase.com
- For more information about setting up messaging in a server farm environment, see "Run the MobiLink server in a server farm" [*MobiLink - Server Administration*].

Starting the MobiLink server for JMS integration

To exchange messages with an external messaging system that supports a JMS interface, you must start the MobiLink server (mlsrv12) with the following options:

- **-c *connection-string*** To connect to the server message store.

  See "-c mlsrv12 option" [*MobiLink - Server Administration*].

- **-m** To enable QAnywhere messaging.

- **-sl java (-cp "*jarfile.jar*")** To add the client JAR files required to use the external JMS provider.

  See "-sl java mlsrv12 option" [*MobiLink - Server Administration*].

Example

The following example starts a MobiLink server using a JMS client library called *jmsclient.jar* (in the current working directory) and the QAnywhere sample database as a message store. The command should be entered all on one line.

```
mlsrv12 -sl java(-cp
"jmsclient.jar") -m -c "dsn=QAnywhere 12 Demo" ...
```

JMS connector property configuration

You use JMS connector properties to specify connection information with the JMS system. They configure a connector to a third party JMS messaging system such as BEA WebLogic or Sybase EAServer.

You can set and/or view properties in several places:

- Sybase Central **Connector Wizard**.

  See "Setting up JMS connectors" on page 129.

- Sybase Central **Connector Properties** window.

- Server management requests.

  See "Creating and configuring connectors" on page 153.

- The ml_qa_global_props MobiLink system table.

For a list of all the JMS connector properties, see "JMS connector properties" on page 726.

Multiple connector configuration

QAnywhere can connect to multiple JMS message systems by defining a JMS connector for each JMS system. The only property value that must be unique among the configured connectors is ianywhere.connector.address.

The ianywhere.connector.address property is the address prefix that QAnywhere clients must specify to address messages meant for the JMS system.

See also

- "Sending a QAnywhere message to a JMS connector" on page 133
- "JMS connector property configuration" on page 132
- "Creating and configuring connectors" on page 153

Sending a QAnywhere message to a JMS connector

A QAnywhere client can send a message to a JMS system by setting the address to the following value:

connector-address\JMS-queue-name

The *connector-address* is the value of the connector property ianywhere.connector.address, while *JMS-queue-name* is the name used to look up the JMS queue or topic using the Java Naming and Directory Interface.

If your *JMS-queue-name* contains a backslash, you must escape the backslash with another backslash. For example, a queue called qq in the context ss should be specified as `ss\\qq`.

```
// C# example
QAMessage msg;
QAManager mgr;
...
mgr.PutMessage( @"ianywhere.connector.wsmqfs\ss\\qq",msg);

// C++ example
QAManagerBase *mgr;
QATextMessage *msg;
...
mgr->putMessage("ianywhere.connector.easerver\\ss\\\\qq", msg);
```

Example

For example, if the ianywhere.connector.address is set to ianywhere.connector.easerver and the JMS queue name is myqueue, then the code to set the address would be:

```
// C# example
QAManagerBase mgr;
QAMessage msg;

// Initialize the manager.
...
msg = mgr.CreateTextMessage()

 // Set the message content.
...
mgr.PutMessage(@"ianywhere.connector.easerver\myqueue", msg);

// C++ example
QAManagerBase *mgr;
QATextMessage *msg;

// Initialize the manager.
...
msg = mgr.createTextMessage();

// Set the message content.
...
mgr->putMessage("ianywhere.connector.easerver\\myqueue", msg);
```

See also

- “QAnywhere message addresses” on page 57
- “JMS connector property configuration” on page 132

Sending a message from a JMS connector to a QAnywhere client

QAnywhere messages are mapped naturally on to JMS messages.

QAnywhere message content

QAnywhere	JMS	Remarks
QATextMessage	javax.jms.TextMessage	message text copied as Unicode
QABinaryMessage	javax.jms.BytesMessage	message bytes copied exactly

QAnywhere built-in headers

The following table describes the mapping of built-in headers. In C++ and JMS, these are method names; for example, Address is called getAddress() or setAddress() for QAnywhere, and getJMSDestination() or setJMSDestination() for JMS. In .NET, these are properties with the exact name given below; for example, Address is Address.

QAnywhere	JMS	Remarks
Address	JMSDestination and JMS property ias_ToAddress	If the destination contains a backslash, you must escape it with a second backslash. Only the JMS part of the address is mapped to the Destination. Under rare circumstances, there may be an additional QAnywhere address suffix if a message loops back into QAnywhere. This is put in ias_ToAddress.
Expiration	JMSExpiration	
InReplyToID	N/A	Not mapped.
MessageID	N/A	Not mapped.
Priority	JMSPriority	
Redelivered	N/A	Not mapped.
ReplyToAddress	JMS property ias_ReplyToAddress	Mapped to JMS property.

QAnywhere	JMS	Remarks
Connector's xjms.receiveDestination property value	JMSReplyTo	ReplyTo set to Destination used by connector to receive JMS messages.
Timestamp	N/A	Not mapped.
N/A	JMSTimestamp	When mapping a JMS message to a QAnywhere message, the JMSTimestamp property of the QAnywhere message is set to the JMSTimestamp of the JMS message.
Timestamp	N/A	When mapping a QAnywhere message to a JMS message, the JMSTimestamp of the JMS message is set to the time of creation of the JMS message.

QAnywhere properties

QAnywhere properties are all mapped naturally to JMS properties, preserving type, with the following exception: if the QAnywhere message has a property called JMSType, then this is mapped to the JMS header property JMSType.

Addressing JMS messages meant for QAnywhere

A JMS client can send a message to a QAnywhere client by setting the JMS message property ias_ToAddress to the QAnywhere address, and then sending the message to the JMS Destination corresponding to the connector property xjms.receiveDestination.

See “QAnywhere message addresses” on page 57.

Example

For example, to send a message to the QAnywhere address "qaddr" (where the connector setting of xjms.receiveDestination is "qanywhere_receive"):

```
import javax.jms.*;
try {
    QueueSession session;
    QueueSender sender;
    TextMessage mgr;
    Queue connectorQueue;

    // Initialize the session.
    connectorQueue = session.createQueue("qanywhere_receive");
```

```
        sender = session.createSender( connectorQueue );
        msg = session.createTextMessage();
        msg.setStringProperty("ias_ToAddress", "qaddr");

        // Set the message content.
        sender.send(msg);
    }
    catch( JMSException e ) {
        // Handle the exception.
    }
```

Mapping JMS messages on to QAnywhere messages

JMS messages are mapped naturally on to QAnywhere messages.

JMS message content

JMS	QAnywhere	Remarks
javax.jms.TextMessage	QATextMessage	Message text copied as Unicode
javax.jms.BytesMessage	QABinaryMessage	Message bytes copied exactly
javax.jms.StreamMessage	N/A	Not supported
javax.jms.MapMessage	N/A	Not supported
javax.jms.ObjectMessage	N/A	Not supported

JMS built-in headers

The following table describes the mapping of built-in headers. In C++ and JMS, these are method names; for example, Address is called getAddress() or setAddress() for QAnywhere, and getJMSDestination() or setJMSDestination() for JMS. In .NET, these are properties with the exact name given below; for example, Address is Address.

JMS	QAnywhere	Remarks
JMS Destination	N/A	The JMS destination must be set to the queue specified in the connector property xjms.receiveDestination.
JMS Expiration	Expiration	
JMS CorrelationID	InReplyToID	
JMS MessageID	N/A	Not mapped.
JMS Priority	Priority	

JMS	QAnywhere	Remarks
JMS Redelivered	N/A	Not mapped.
JMS ReplyTo and connector's ianywhere.connector.address property value	ReplyToAddress	The connector address is concatenated with the JMS ReplyTo Destination name delimited by '\'.
JMS DeliveryMode	N/A	Not mapped.
JMS Type	QAnywhere message property JMSType	
JMS Timestamp	N/A	Not mapped.

JMS properties

JMS properties are all mapped naturally to QAnywhere properties, preserving type, with a few exceptions. The QAnywhere Address property is set from the value of the JMS message property ias_ToAddress. If the JMS message property ias_ReplyToAddress is set, then the QAnywhere ReplyToAddress is additionally suffixed with this value delimited by a '\'.

Web service connectors

A web service connector listens for QAnywhere messages sent to a particular address, and makes web service calls when messages arrive. Web service responses are sent back to the originating client as QAnywhere messages. All messages sent to the web services connector should be created using the proxy classes generated by the QAnywhere WSDL compiler.

When running MobiLink with QAnywhere messaging in a server farm environment, only the QAnywhere connectors in the primary server start. If the primary server fails, the QAnywhere connectors are automatically started in the new primary server so that message ordering is preserved while exchanging data in the external messaging system, such as JMS. For more information, see "Run the MobiLink server in a server farm" [*MobiLink - Server Administration*].

Setting up web service connectors

Create a web service connector

1. Open Sybase Central and connect to your server message store.
2. Click **File** » **New** » **Connector**.
3. Click **Web Services**. Click **Next**.

4. In the **Connector Name** field, type the connector address that a QAnywhere client should use to address the connector. Click **Next**.

5. In the **URL** field, type the URL of the web service (for example, *http://localhost:8080/qanyserv/F2C*). Click **Next**.

   You can optionally specify a timeout period in milliseconds, which cancels requests if the web service does not respond in the specified time. This sets the property webservice.socket.timeout.

6. On the **HTTP Parameters** page, click **The Web Service Uses HTTP Authentication** and complete the following fields:

   - In the **HTTP user name** field, type the user name. This sets the property webservice.http.authName.
   - In the **HTTP password** field, type the password. This sets the property webservice.http.password.e.

7. Click **The Web Service Must Be Accessed Through A Proxy** and complete the following fields:

   - In the **Proxy host name** field, type the host name. If you specify this property, you must specify the webservice.http.proxy.port property.
   - In the **Proxy port** field, type the port to connect to on the proxy server. If you specify this property, you must specify the webservice.http.proxy.host property.
   - In the **Proxy user name** field, type the proxy user name to use if the proxy requires authentication. If you specify this property, you must also specify the webservice.http.proxy.password.e property.
   - Click **Finish**.

   To set additional options on your web service connector, you can right-click the connector you just created and click **Properties**; or you can use server management requests.

   For a list of available properties, see "Web service connector properties" on page 138.

   For information about using server management requests, see "Connector administration with server management requests" on page 152.

Web service connector properties

Use web service connector properties to specify connection information with the web service. You can set these properties in the Sybase Central Connector Wizard.

See "Web service connectors" on page 137.

You can view web service connector properties in the Sybase Central **Connector Properties** window, or in the ml_qa_global_props MobiLink system table.

To open the **Connector Properties** window, right-click the connector in Sybase Central and click **Properties**.

View web service properties

1. Open Sybase Central and connect to your server message store.
2. Under **Server Message Stores** in the left pane, click the name of your data source.
3. In the right pane, click the **Connectors** tab, and then click the name of the web service connector.
4. Click **File** » **Properties**

Web service connector properties

- **ianywhere.connector.nativeConnection** The Java class that implements the connector. It is for QAnywhere internal use only, and should not be deleted or modified.
- **ianywhere.connector.id (deprecated)** An identifier that uniquely identifies the connector. The default is ianywhere.connector.address.
- **ianywhere.connector.address** The connector address that a QAnywhere client should use to address the connector. This address is also used to prefix all logged error, warning, and informational messages appearing in the MobiLink messages window for this connector.

  In Sybase Central, you set this property in the **Connector Wizard**, **Connector Name** page, **Connector Name** field.
- **ianywhere.connector.compressionLevel** The default compression factor of messages received from the web service. Compression is an integer between 0 and 9, with 0 indicating no compression and 9 indicating maximum compression.

  In Sybase Central, you set this property on the **Connector Properties** window, on the **General** tab, in the **Compression Level** section.
- **ianywhere.connector.logLevel** The amount of connector information displayed in the MobiLink messages window and the MobiLink server message log file. Values for the log level are as follows:
  - **1** Log error messages.
  - **2** Log error and warning messages.
  - **3** Log error, warning, and information messages.
  - **4** Log error, warning, information, and debug messages.

  In Sybase Central, you set this property on the **Connector Properties** window, on the **General** tab, in the **Logging Level** section.
- **ianywhere.connector.outgoing.retry.max** The default number of retries for messages going from QAnywhere to the external messaging system. The default value is 5. Specify 0 to have the connector retry forever.

  In Sybase Central, you can set this property in the **Connector Properties** window under the **Properties** tab by clicking **New**.

- **ianywhere.connector.startupType** Startup types can be automatic, manual, or disabled.
- **webservice.http.authName** If the web service requires HTTP authentication, use this property to specify the user name.
- **webservice.http.password.e** If the web service requires HTTP authentication, use this property to specify the password.
- **webservice.http.proxy.authName** If the proxy requires authentication, use this property to set the proxy user name. If you specify this property, you must also specify the webservice.http.proxy.password.e property.
- **webservice.http.proxy.host** If the web service must be accessed through an HTTP proxy, use this property to specify the host name. If you specify this property, you must specify the webservice.http.proxy.port property.
- **webservice.http.proxy.password.e** If the proxy requires authentication, use this property to set the proxy password. If you specify this property, you must also specify the webservice.http.proxy.authName property.
- **webservice.http.proxy.port** The port to connect to on the proxy server. If you specify this property, you must specify the webservice.http.proxy.host property.

Sending a message to a web service connector

A message is sent to a web service connector through the mobile web services API. Use the setProperty method from the class ianywhere.qanywhere.ws.WSBase to set the WS_CONNECTOR_ADDRESS property to the ID of the web service connector. See "WSBase class [QAnywhere Java]" on page 586.

For example, when the following line of code from the CurrencyConvertor sample is specified, the web service APIs used to make web service requests send these requests as messages through the web service connector.

```
service.setProperty(
    "WS_CONNECTOR_ADDRESS",
    "ianywhere.connector.currencyconvertor\\" );
```

Tutorial: Using JMS connectors

A JMS connector provides connectivity between a JMS message system and QAnywhere. In this tutorial, you send messages between a Windows JMS client application and a QAnywhere client application.

Required software

- SQL Anywhere 12
- Java Software Development Kit
- A JMS system

Competencies and experience

You require:

- Familiarity with Java
- Basic knowledge of configuring your JMS connector

Goals

You gain competence and familiarity with:

- Configuring your JMS connector to communicate with a sample QAnywhere application
- Sending messages between a JMS message system and a sample QAnywhere application

Key concepts

This section uses the following steps to provide connectivity between a JMS message system and QAnywhere using a SQL Anywhere sample database:

- Preparing your JMS connector to send and receive messages
- Running the QAnywhere server and client components, and the JMS client
- Sending a message from the QAnywhere client to the JMS client, and vice-versa

Suggested background reading

For more information about using JMS connectors, see "Connectors" on page 129.

Lesson 1: Setting up client and server components

Prepare your JMS provider

1. Refer to your JMS server documentation to start the server.
2. Create the following queues within your JMS server:
   - **testmessage** The TestMessage sample uses this queue name to listen for messages.
   - **qanywhere_receive** The QAnywhere JMS connector uses this queue name to listen for messages.

   You may need to restart the server after creating the queues. Refer to your JMS server documentation for more details.

Start the QAnywhere client and server components

1. Create a QAnywhere JMS connector for your JMS system using Sybase Central. See "Setting up JMS connectors" on page 129.
2. At a command prompt, run the following command:

```
mlsrv12 -m -c "dsn=QAnywhere 12 Demo" -sl
java(-cp JMS-client-jar-files) -vcrs
-zu+
```

where *JMS-client-jar-files* is a semicolon delimited list of jar files that are required to access the JMS server. See your JMS server documentation for details.

The MobiLink server starts for messaging.

3. Click **Start** » **Programs** » **SQL Anywhere 12** » **QAnywhere** » **Tutorial using SQL Anywhere** » **QAnywhere Agent For SQL Anywhere -- saclient1**.

   The QAnywhere Agent loads.

4. Click **Start** » **Programs** » **SQL Anywhere 12** » **QAnywhere** » **Tutorial using SQL Anywhere** » **TestMessage -- saclient1**.

   The QAnywhere sample application loads.

Start the JMS version of the TestMessage client

1. At a command prompt, run the following command:

   ```
   edit "%SQLANYSAMP12%/QAnywhere/JMS/TestMessage/build.bat"
   ```

2. Examine the code in the *build.bat* file and ensure that your JMS server file paths are correct.

   For example, if you use EAServer, the default settings are defined under the **easerver** heading:

   ```
   :easerver
   REM For EAServer, compile with the following JAR files
   SET easerver_install=c:\program files\sybase\easerver6
   SET jmsjars=%easerver_install%\lib\eas-client-15.jar
   GOTO build_app
   ```

   If EAServer is not located in the *c:\program files\sybase\easerver6* directory, update the **easerver_install** variable so that it points to the proper install directory. Make sure that the **jmsjars** variable points to the proper location of the JMS server jar files.

   If your JMS server is not listed, use the **custom** header settings defined near the beginning of the batch file to define your own JMS file path locations.

   When finished, save your changes and exit the editor.

3. At a command prompt, run the following command to compile the JMS TestMessage client:

   ```
   build.bat JMS-server-name
   ```

   where *JMS-server-name* is the name of your JMS server represented as a header name in *build.bat*. Acceptable values are **easerver**, **fioranomq**, **jboss**, **tibco**, **weblogic**, and **custom**. By default, *build.bat* uses **easerver**.

4. At a command prompt, run the following command:

   ```
   edit "%SQLANYSAMP12%/QAnywhere/JMS/TestMessage/run.bat"
   ```


5. Examine the code in the *run.bat* file and ensure that your JMS server file paths are correct.

   For example, if you use EAServer, the default settings are defined under the **easerver** heading:

   ```
   :easerver
   REM For EAServer, compile with the following JAR files
   SET easerver_install=c:\program files\sybase\easerver6
   SET jmsjars=%easerver_install%\lib\eas-client-15.jar
   GOTO build_app
   ```

   If EAServer is not located in the *c:\program files\sybase\easerver6* directory, update the **easerver_install** variable so that it points to the proper install directory. Make sure that the **jmsjars** variable points to the proper location of the JMS server jar files.

   If your JMS server is not listed, use the **custom** header settings defined near the beginning of the batch file to define your own JMS file path locations.

   When finished, save your changes and exit the editor.

6. At a command prompt, run the following command to run the JMS TestMessage client:

   ```
   run.bat JMS-server-name
   ```

   where *JMS-server-name* is the name of your JMS server represented as a header name in *build.bat*. Acceptable values are **easerver**, **fioranomq**, **jboss**, **tibco**, **weblogic**, and **custom**. By default, *build.bat* uses **easerver**.

7. Move the JMS TestMessage window to the right side of your screen under the existing **saclient1 - TestMessage** window.

8. Proceed to "Lesson 2: Sending a message from a JMS client to a QAnywhere client".

Lesson 2: Sending a message from a JMS client to a QAnywhere client

This lesson assumes you have completed all preceding lessons. See "Lesson 1: Setting up client and server components".

Send a message from a JMS client to a QAnywhere client

1. From JMS TestMessage **Message** menu, click **New**.

2. In the **Destination ID** field, type **saclient1**.

3. Complete the **Subject** and **Message** fields with sample text.

4. Click **Send**.

   A window appears, indicating that a message has been received.

5. Proceed to "Lesson 3: Sending a message from a QAnywhere client to a JMS client".

Lesson 3: Sending a message from a QAnywhere client to a JMS client

This lesson assumes you have completed all preceding lessons. See "Lesson 1: Setting up client and server components".

Find out the name of the QAnywhere JMS connector

1. Click **Start** » **Programs** » **SQL Anywhere** » **Sybase Central**.
2. Click **Connections** » **Connect With QAnywhere 12**.
3. Click **ODBC Data Source Name**.
4. Click **Browse** and click **QAnywhere 12 Demo**.
5. Click **OK**.
6. Click **OK**.
7. Click the **Connectors** tab.

   The right-pane displays a list of all active JMS connectors.
8. Examine the name field.

   There should only be one active QAnywhere JMS connector in the list. The name of the connector is displayed under the name field.

Send a message from a QAnywhere client to a JMS client

1. In the **saclient1 - TestMessage** window, click **Message** » **New**.
2. In the **Destination ID** field, type the name of your JMS system.
3. In the **Subject** and **Message** fields, type sample text.
4. Click **Send**.

   A window appears, indicating that a message has been received.
5. Proceed to "Cleaning up".

Cleaning up

This lesson assumes you have completed all preceding lessons. See "Lesson 1: Setting up client and server components".

Shut down TestMessage clients, the QAnywhere Agent, and the MobiLink server.

Shut down all applications

1. To close the JMS TestMessage client application, click **File** » **Exit**.
2. To close the **saclient1 - TestMessage Client** window, click **File** » **Exit**.
3. To close the **saclient1 - QAnywhere Agent** window and the MobiLink server, click **Shut Down** in their respective windows.
4. To disconnect from your JMS connector, refer to your JMS server documentation.

Server management requests

A QAnywhere client application can send special messages to the server called **server management requests**. These messages contain content that is formatted as XML and are addressed to the QAnywhere system queue. They require a special authentication string. Server management requests can perform a variety of functions, such as:

- Starting and stopping connectors and web services.

  See "Opening connectors" on page 155 and "Closing connectors" on page 155.

- Scheduling server management requests.

  See "Schedule tag" on page 665.

- Monitoring connector status.

  See "Connector monitoring" on page 156.

- Setting and refreshing client transmission rules.

  See "Transmission rule specification with a server management request" on page 162.

- Monitoring message status.

  See "QAnywhere monitoring" on page 165.

- Setting, updating, deleting, and querying client message store properties on the server.

  See "Server property setup with a server management request" on page 161.

- Canceling messages.

  See "Message cancellation" on page 151.

- Querying for active clients, message store properties, and messages.

Addressing server management requests

By default, server management requests must be addressed to **ianywhere.server\system**. To change the client ID portion of this address, set the ianywhere.qa.server.id property and restart the server. For example, if the ianywhere.qa.server.id property is set to myServer, server management requests are addressed to myServer\system.

For more information about setting the ianywhere.qa.server.id property, see "Server properties" on page 724.

For more information about addressing QAnywhere messages, see "Sending QAnywhere messages" on page 61.

For more information about the system queue, see "System queue" on page 58.

Examples

The following is a sample message details request. It generates a single report that displays the message ID, status, and target address of all messages with priority 9 currently on the server.

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
   <MessageDetailsRequest>
      <request>
         <requestId>testRequest</requestId>
         <condition>
            <priority>9</priority>
         </condition>
         <status/>
         <address/>
      </request>
   </MessageDetailsRequest>
</actions>
```

The following example is in C#. It sets a server-side transmission rule for a client such that messages from the server are only transmitted to the client called someClient if the priority is greater than 4.

```
QAManager mgr = ...; // Initialize the QAManager
QAMessage msg = mgr.CreateTextMessage();
msg.SetStringProperty( "ias_ServerPassword", "QAnywhere" );

// Indenting and newlines are just for readability
msg.Text = "<?xml version="1.0" encoding="UTF-8"?>\n"
+ "<actions>\n"
+ " <SetProperty>\n"
+ "  <prop>\n"
+ "   <client>someClient</client>\n"
+ "   <name>ianywhere.qa.server.rules</name>\n"
+ "   <value>ias_Priority &gt; 4</value>\n"
+ "  </prop>\n"
+ " </SetProperty>\n"
+ " <RestartRules>\n"
+ "  <client>someClient</client>\n"
+ " </RestartRules>\n"
+ "</actions>\n";

mgr.PutMessage( @"ianywhere.server\system", msg );
```

Authenticating server management requests

The **ianywhere.qa.server.password.e** server property is used to specify a password that is used for authenticating server management requests. If this property is not set, the password is QAnywhere. See "Server properties" on page 724.

Server management request writing

Server management requests contain content that is formatted as XML.

Note
You cannot use symbols such as > or < in the content of server management requests. Instead, use > and <.

Each type of server management request includes it own XML tags. For example, to close a connector you use the <CloseConnector> tag.

Each server management request starts with an <actions> tag.

actionsResponseId Tag

Use the actionsResponseId tag as a subtag to the <actions> tag to track and report the progress of the operations in the <action> tag. A report is created by the server when the <action> tag is processed.

The report contains the id of the <actionsResponseId> tag and the error messages generated by the request. Once the report is created, it is sent to the reply address of the server management request.

The following is an example of a server management request using the actionsResponseId tag,

```
<?xml version="1.0" encountered="UTF-8"?>
<actions>
  <actionsResponseId>myActionID</actionsResponseId>
  <MessageDetailsRequest>
    <request>
      <requestId>testRequest</requestId>
      <condition>
        <priority>9</priority>
      </condition>
      <status/>
      <address/>
    </request>
  </MessageDetailsRequest>
</actions>
```

The following is an example of an actionsResponseId report where the myActionId request did not generate errors.

```
<?xml version="1.0" encoding="UTF-8"?>
<ActionsResponse>
<actionsResponseId>myActionId</actionsResponseId>
<error/>
</ActionsResponse>
```

Archive message store requests

To view the details of messages in the archive message store, use the <archived> tag as a subtag to the <condition> tag. If the tag is omitted, the report only contains messages from the server message store.

To determine if a message exists in the archive message store, use the <archived> tag as a subtag to the <request> tag.

Example

The following request returns true if testRequest exists in the archive message store, and false if it exists in the server message store.

```
<request>
   <requestID>testRequest</requestID>
```

```
    <status/>
    <archived/>
</request>
```

Destination alias creation

You can use server management requests to create and modify destination aliases.

See also

- "Destination alias creating with a server management request" on page 163
- "Destination aliases" on page 127.
- "Server management requests" on page 147

Server message store administration with server management requests

You can use server management requests to administer the server message store.

See also

- "Server management requests" on page 147

Client transmission rule refresh

When a server-side client transmission rule is changed, the rules for the corresponding client must be refreshed. You can change client transmission rules in a server management request by setting the property ianywhere.qa.server.rules.

A RestartRules tag contains a single client tag, which specifies the name of the client to refresh.

<RestartRules> subtags	Description
<client>	The name of the client for which to refresh transmission rules.

Example

The server XML needs to specify the new transmission rule property and then restart rule processing using the RestartRules tag. For example, the following XML changes the server-side transmission rule for client myclient to `auto = ias_Priority > 4`. Note the proper encoding of ">" in the XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
    <SetProperty>
      <prop>
        <client>myclient</client>
```



```
            <name>ianywhere.qa.server.rules</name>
            <value>auto = ias_Priority &gt; 4</value>
          </prop>
        </SetProperty>
        <RestartRules>
            <client>myclient</client>
        </RestartRules>
    </actions>
```

Message cancellation

You can create a server management request to cancel messages in the server message store. You can create a one-time cancellation request or you can schedule your cancellation request to happen automatically. You can also optionally generate a report that details the messages that have been canceled.

Messages can only be canceled if they are in a non-final state and have not been transmitted to the recipient when the request is activated.

<CancelMessageRequest> subtags	**Description**
<request>	Groups information about a particular request. Specifying more than one <request> tag is equivalent to sending multiple separate server management requests.

<Request> subtags	**Description**
<condition>	Groups conditions for including a message to be canceled. See "Condition tag" on page 664.
<persistent>	Specifies that the request should be made persistent in the server database (so that messages can be canceled even if the server is restarted). Only used with schedules.
<requestId>	Specifies a unique identifier for the request that is included in each report generated as a result of this request. Using different values for this field allows more than one request to be active at the same time. Using the same request id allows the client to override or delete active requests.
<replyAddr>	The return address for each report generated as a result of this request. If this tag is omitted, the default return address of reports is the return address of the originating message.
<report>	Causes a report to be sent each time the request is activated. To cause a report to be sent each time the request is activated, put an empty <report> tag inside the <request> tag.

<Request> subtags	Description
<schedule>	Specifies that the report should be generated on a schedule. See "Server management request parent tags" on page 664.

Example

This request cancels messages on the server with the address ianywhere.connector.myConnector \deadqueue:

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
 <CancelMessageRequest>
  <request>
   <requestId>cancelRequest</requestId>
   <condition>
    <customRule>ias_Address='ianywhere.connector.myConnector\deadqueue'</
customRule>
   </condition>
  </request>
 </CancelMessageRequest>
</actions>
```

Message deletion

To specify a clean-up policy on the server, set the property ianywhere.qa.server.deleteRules for the special client ianywhere.server.deleteRules with the rule or rules governing which messages can be deleted from the server.

The following example changes the message clean-up policy to delete expired and canceled messages:

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
    <SetProperty>
 <prop>
     <client>ianywhere.server.deleteRules</client>
     <name>ianywhere.qa.server.deleteRules</name>
     <value>auto = ias_Status in ( ias_ExpiredStatus, ias_CancelledStatus )
and ias_TransmissionStatus = IAS_TRANSMITTED</value>
 </prop>
    </SetProperty>
    <RestartRules>
 <client>ianywhere.server.deleteRules</client>
    </RestartRules>
</actions>
```

Connector administration with server management requests

You can use server management requests to create, configure, delete, start, stop, and monitor connectors.

See also

- "Server management requests" on page 147
- "Connectors" on page 129
- "Web service connectors" on page 137

Creating and configuring connectors

To create connectors, add properties using <SetProperty> and then use <OpenConnector>.

Example

In the following example, the server management request first sets several relevant properties and associates them with the client ianywhere.connector.jboss, which is the client ID of the new connector. JMS-specific properties are set in such a way that a connector to a local JBOSS JMS server are indicated. The connector is then started using the OpenConnector tag. Note that if you have not started the MobiLink server with the relevant jar files of the JMS client, the connector is not started.

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
    <SetProperty>
        <prop>
       <client>ianywhere.connector.jboss</client>
       <name>ianywhere.connector.nativeConnection</name>
  <value>ianywhere.message.connector.jms.NativeConnectionJMS</value>
   </prop>
        <prop>
       <client>ianywhere.connector.jboss</client>
       <name>ianywhere.connector.address</name>
  <value>ianywhere.connector.jboss</value>
   </prop>
        <prop>
       <client>ianywhere.connector.jboss</client>
       <name>xjms.jndi.factory</name>
  <value>org.jnp.interfaces.NamingContextFactory</value>
   </prop>
        <prop>
       <client>ianywhere.connector.jboss</client>
       <name>xjms.jndi.url</name>
  <value>jnp://0.0.0.0:1099</value>
   </prop>
        <prop>
       <client>ianywhere.connector.jboss</client>
       <name>xjms.topicFactory</name>
  <value>ConnectionFactory</value>
   </prop>
        <prop>
       <client>ianywhere.connector.jboss</client>
       <name>xjms.queueFactory</name>
  <value>ConnectionFactory</value>
   </prop>
        <prop>
       <client>ianywhere.connector.jboss</client>
       <name>xjms.receiveDestination</name>
  <value>qanywhere_receive</value>
   </prop>
        <prop>
       <client>ianywhere.connector.jboss</client>
```

```
        <name>xjms.deadMessageDestination</name>
  <value>qanywhere_deadMessage</value>
   </prop>
    </SetProperty>
    <OpenConnector>
        <client>ianywhere.connector.jboss</client>
    </OpenConnector>
</actions>
```

Modifying connectors

To modify connectors, close the connector, change properties with the <SetProperty> tag, and then open the connector.

Example

In the following example, the logging level of the connector is changed to 4. The connector with the ID ianywhere.connector.jboss is closed; the connector property logLevel is changed to 4, and then the connector is re-opened with the new log level.

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
    <CloseConnector>
        <client>ianywhere.connector.jboss</client>
    </CloseConnector>
    <SetProperty>
        <prop>
       <client>ianywhere.connector.jboss</client>
  <name>ianywhere.connector.logLevel</name>
  <value>4</value>
   </prop>
    </SetProperty>
    <OpenConnector>
        <client>ianywhere.connector.jboss</client>
    </OpenConnector>
</actions>
```

Deleting connectors

To delete connectors, use <SetProperty> to remove all properties for the client.

Example

In the following example, the connector with the ID ianywhere.connector.jboss is closed. All of its properties are deleted by the <SetProperty> tag, omitting the name and value tags.

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
    <SetProperty>
        <prop>
       <client>ianywhere.connector.jboss</client>
       <name>ianywhere.connector.nativeConnection</name>
     </prop>
    </SetProperty>
</actions>
```


Opening connectors

To open connectors, use <OpenConnector>.

An OpenConnector tag contains a single client tag that specifies the name of the connector to open.

<OpenConnector> subtag	Description
<client>	The name of the connector to open.

See also

- "Connectors" on page 129
- "Web service connectors" on page 137

Example

The following example opens the simpleGroup connector.

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
    <OpenConnector>
     <client>simpleGroup</client>
    </OpenConnector>
</actions>
```

Closing connectors

To close connectors, use <CloseConnector>. A CloseConnector tag contains a single client tag that specifies the name of the connector to close.

<CloseConnector> subtags	Description
<client>	The name of the connector to close.

See also

- "Connectors" on page 129
- "Web service connectors" on page 137

Example

The following example closes the simpleGroup connector.

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
    <CloseConnector>
     <client>simpleGroup</client>
    </CloseConnector>
</actions>
```

Connector monitoring

To obtain information about connectors, write a special kind of server management request called a client status request. It contains a <ClientStatusRequest> tag that uses one or more <request> tags containing the information necessary to register the request.

Your client status request can obtain reports about connectors in several ways:

- Make a one-time request.
- Register a State Change Listener to have a report sent whenever the connector's state changes.
- Register an Error Listener to have a report sent whenever an error occurs on the connector.

In addition, you can schedule a report to be sent at certain times or intervals.

ClientStatusRequest tag

To get information about connectors, use <ClientStatusRequest>.

A client status request is composed of one or more <request> tags containing all the necessary information to register the request.

<ClientStatusRequest> subtag	Description
<request>	Groups information in requests.

request tag for client status requests

In the <request> tag, use an optional <replyAddr> tag to specify the return address for each report generated as a result of this request. If this tag is omitted, the default return address of reports is the reply address of the originating message.

Use an optional <requestId> to add a label for the request that is included in each report. When you register multiple requests, or when you delete or modify requests, the ID makes it possible to distinguish which reports were generated from a particular request.

To specify a list of connectors for the request, include one or more <client> tags, each with one connector address. For a one-time request, all the connectors are included in the report. For an event listener request, the server listens to each of these connectors.

To specify that event details should be made persistent during any server downtime, specify the <persistent> tag. This tag does not require any data and can be of the form <persistent/> or <persistent></persistent>.

You can optionally specify a list of events by including one or more <onEvent> tags with one event type per tag. If these tags are omitted, the client status request produces a one-time request. Otherwise, the client status request registers event listeners for the specified events.

<request> subtags for client status requests	Description
<client>	You can include one or more <client> tags, with one connector address per tag. For a one-time request, all the connectors listed are included in the report. For an event listener request, the server begins to listen to each of these connectors.
<onEvent>	Specifies the events upon which the server should generate reports. You can include one or more <onEvent> tags, with one event type per tag. If these tags are omitted, the Client Status Request produces a one-time request. Otherwise, the Client Status Request is used to register event listeners for the specified events.
<persistent>	Specifies that the details information in this Client Status Request should be made persistent in the server database.
<replyAddr>	Specifies the return address for each report generated as a result of this request. If this tag is omitted, the default return address of reports is the reply address of the originating message.
<requestId>	A label for the report. This value is used as a label for the request and is included in each report generated as a result of this request. This makes it possible to distinguish which reports were generated from a particular request when multiple requests have been registered and to delete or modify outstanding requests.
<schedule>	See "Server management request parent tags".

One-time client status requests

You create a one-time request by omitting <onEvent> and <schedule> tags from the client status request. In this case, a single report is generated that contains the current status information for each connector specified in the client status request.

The following XML message omits the <onEvent> and <schedule> tags and so is an example of a one-time request. It generates a single report containing the current status information for each connector specified in the <ClientStatusRequest> tag.

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
 <ClientStatusRequest>
  <request>
   <replyAddr>ianywhere.connector.beajms\q11</replyAddr>
   <requestId>myOneTimeRequest</requestId>
   <client>ianywhere.server</client>
   <client>ianywhere.connector.beajms</client>
  </request>
 </ClientStatusRequest>
</actions>
```

On-event client status requests

To specify events for which you want the QAnywhere Server to generate status reports, include one or more <onEvent> tags in your client status request. Unlike one-time requests, the server does not immediately respond to the request, but instead begins listening for events to occur. Each time one of these events is triggered, a report is sent containing information about the connector that caused the event.

The following events are supported for on-event requests:

Event	When it occurs
open	A closed connector is opened.
close	A previously opened or paused connector is closed.
statusChange	The status of the connector is changed from one state to another. Possible states are open and close.
error	An unexpected error is thrown by the connector.
fatalError	An unhandled fatal error is thrown by the connector.
none	This never occurs. This effectively removes all previous event watches from the connector.

In the following example, the connector with address ianywhere.connector.beajms\q11 is sent a status report each time the server connector changes its status or generates an error.

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
 <ClientStatusRequest>
  <request>
   <replyAddr>ianywhere.connector.beajms\q11</replyAddr>
   <requestId>myEventRequest</requestId>
   <client>ianywhere.server</client>
   <onEvent>statusChange</onEvent>
   <onEvent>error</onEvent>
  </request>
 </ClientStatusRequest>
</actions>
```

Multiple simultaneous requests

Each return address can have its own set of event listeners for any number of connectors, including the server connector. Adding an event listener to a connector does not disturb any other event listeners in the server (except possibly one that it is replacing).

Request replacement

If you add an event listener to a connector that already has an event listener registered to it by the same return address, it replaces the old listener with the new one. For example, if a statusChange listener for connector abc is registered to address x/y and you register an error listener for abc to address x/y, abc no longer responds to statusChange events.

To register more than one event to the same address, you must create a single request with more than one <onEvent> tag.

Removing a request

If an event listener for a connector is registered to an address, you can remove the event listener by providing another client status request from the same address with the "none" event specified.

In the following example, all event listeners are removed for the server connector registered to the address ianywhere.connector.beajms\q11:

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
 <ClientStatusRequest>
  <request>
   <replyAddr>ianywhere.connector.beajms\q11</replyAddr>
   <client>ianywhere.server</client>
   <onEvent>none</onEvent>
  </request>
 </ClientStatusRequest>
</actions>
```

Persistent client status requests

To specify that the details of a request are saved into the global properties table on the message store (where they can be automatically reinstated after a server restart), include the <persistent> tag in a client status request. Persistence can be used with scheduled events and event listeners, but not one-time requests. The rules for adding and removing persistent requests are similar to those for regular requests, except that scheduled events and event listeners cannot be added separately. Instead, when adding a persistent request, the client must specify all event listeners and schedules for a particular connector/reply address pair in the same request.

The following example adds the event listener and schedule to ianywhere.connector.myConnector and makes them persistent. It also overwrites any previous persistent requests from this connector/reply address pair. A report is sent every half hour and when a connector status change occurs.

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
 <ClientStatusRequest>
  <request>
   <replyAddr>ianywhere.connector.beajms\q11</replyAddr>
   <client>ianywhere.connector.myConnector</client>
   <onEvent>statusChange</onEvent>
   <schedule>
    <everyminute>30</everyminute>
   </schedule>
   <persistent/>
  </request>
 </ClientStatusRequest>
</actions>
```

Event listener persistence

If a connector is closed, any event listeners it has registered to its address persist in the server until the server is shut down. If the connector is reopened, the stored event listeners become active again.

Connector states

A connector can be in one of two states:

- **running** The connector is accepting and processing incoming and outgoing messages. This state is reflected in the connector property ianywhere.connector.state=1.
- **not running** The connector is not accepting or processing incoming or outgoing messages. This state is reflected in the connector property ianywhere.connector.state=2. When the connector state is changed to "running" the connector is initialized from scratch.

See also

- "Modifying connectors" on page 154

Client status reports

A client status report is generated by the server each time a report is requested by a connector or a registered event occurs. It is generated as a simple text message which does not contain any message properties.

Depending on what information is available at the time of the event, any of the following values may be included in each component report:

- client (always present)
- UTCDatetime (always present)
- vendorStatusDescription (always present)
- statusCode (always present)
- vendorStatusCode
- statusSubCode
- statusDescription

For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<ClientStatusReport>
 <requestId>myRequest</requestId>
 <componentReport>
  <client>ianywhere.server</client>
  <UTCDatetime>Tue May 31 13:53:02 EDT 2005</UTCDatetime>
  <statusCode>Running</statusCode>
  <vendorStatusDescription></vendorStatusDescription>
 </componentReport>
 <componentReport>
  <client>ianywhere.connector.beajms</client>
  <UTCDatetime>Tue May 31 13:53:02 EDT 2005</UTCDatetime>
  <statusCode>Not running</statusCode>
  <vendorStatusDescription></vendorStatusDescription>
 </componentReport>
</ClientStatusReport>
```


Server property setup with a server management request

A <SetProperty> tag contains one or more <prop> tags, each of which specifies a property to set. Each prop tag consists of a <client> tag, a <name> tag, and a <value> tag. To delete a property, omit the <value> tag.

<prop> subtags	Description
<client>	The name of the client for which to set a server property.
<name>	The name of the property to set.
<value>	The value of the property being set. If not included, the property gets deleted.

Example

The following server management request sets the ianywhere.qa.member.client3 property to Y for the destination alias called simpleGroup, which adds client3 to simpleGroup.

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
    <SetProperty>
     <prop>
        <client>simpleGroup</client>
        <name>ianywhere.qa.member.client3</name>
        <value>Y</value>
     </prop>
    </SetProperty>
</actions>
```

The next example does the following:

- Creates or modifies the value of the client1 property myProp1 to 3.
- Deletes the client1 property myProp2.
- Modifies the value of the client2 property myProp3 to "some value".

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
    <SetProperty>
 <prop>
     <client>client1</client>
     <name>myProp1</name>
     <value>3</value>
 </prop>
 <prop>
     <client>client1</client>
     <name>myProp2</name>
 </prop>
 <prop>
     <client>client2</client>
     <name>myProp3</name>
```

```
      <value>some value</value>
 </prop>
    </SetProperty>
</actions>
```

See also

- "Server management requests" on page 147

Transmission rule specification with a server management request

With a server management request, you can specify default server transmission rules that apply to all users, or you can specify transmission rules for each client.

To specify default transmission rules (for a server), set the ianywhere.qa.server.rules property for the client ianywhere.server.defaultClient. For a client, use the ianywhere.qa.server.rules property to specify server transmission rules.

For an overview of how to use server management requests, including how to authenticate and schedule them, see "Server management requests" on page 147.

Example

The following example creates the default rule that only high priority messages (priority greater than 6) should be sent:

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
  <SetProperty>
    <prop>
      <client>ianywhere.server.defaultClient</client>
      <name>ianywhere.qa.server.rules</name>
      <value>auto = ias_Priority &gt; 6</value>
    </prop>
  </SetProperty>
  <RestartRules>
    <client>ianywhere.server.defaultClient</client>
  </RestartRules>
</actions>
```

The following example creates a rule for a client called myClient that only messages with a content size less than 100 should be transmitted during business hours (8 a.m. and 6 p.m.):

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
  <SetProperty>
    <prop>
      <client>myClient</client>
      <name>ianywhere.qa.server.rules</name>
      <value>auto = ias_ContentSize &lt; 100
        and ias_CurrentTime &gt; '8:00:00'
        and ias_CurrentTime &lt; '18:00:00'</value>
    </prop>
  </SetProperty>
```



```
  <RestartRules>
    <client>myClient</client>
  </RestartRules>
</actions>
```

Destination alias creating with a server management request

You can use server management requests to create and modify destination aliases.

For more information about destination aliases, see "Destination aliases" on page 127.

To create a destination alias, send a server management request in which the client name is the name of the destination alias and the following properties are specified. The group is identified by the group, address, and nativeConnection properties. Members of the group are specified with the member property.

```
<prop>
  <client>simpleGroup</client>
  <name>ianywhere.connector.nativeConnection</name>
  <value>ianywhere.message.connector.group.GroupConnector
  </value>
</prop>
```

Property	Description

ianywhere.qa.group	Set this property to Y to indicate that you are configuring a destination alias. For example:
	`<prop>`
	`  <client>simpleGroup</client>`
	`  <name>ianywhere.qa.group</name>`
	`  <value>Y</value>`
	`</prop>`

ianywhere.connector.address	Specify the client ID of the destination alias. For example:
	`<prop>`
	`  <client>simpleGroup</client>`
	`  <name>ianywhere.connector.address</name>`
	`  <value>simpleGroup</value>`
	`</prop>`

ianywhere.connector.nativeConnection	Set to ianywhere.message.connector.group.GroupConnector. For example:
	`<prop>`
	`  <client>simpleGroup</client>`
	`  <name>ianywhere.connector.nativeConnection</name>`
	`  <value>ianywhere.message.connector.group.GroupConnector`
	`  </value>`
	`</prop>`

Property	Description

ianywhere.qa.member.*client-name**queue-name*	Specify Y to add a member or N to remove a member. You can also optionally specify a delivery condition. See "Condition syntax" on page 731.
	
	For example, to add client1 to the destination alias simpleGroup, set the property as follows. The queue-name is optional. Repeat this property for every client you want to add:
	
	`<prop>`
	`  <client>simpleGroup</client>`
	`  <name>ianywhere.qa.member.client1\queue1</name>`
	`  <value>Y</value>`
	`</prop>`

See also

- "Server management requests" on page 147
- "QAnywhere transmission and delete rules" on page 729

Example

The following server management request creates a destination alias called simpleGroup with members called client1 and client2\q11. This example starts the destination alias so that it immediately begins handling messages.

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
  <SetProperty>
    <prop>
      <client>simpleGroup</client>
      <name>ianywhere.qa.group</name>
      <value>Y</value>
    </prop>
    <prop>
      <client>simpleGroup</client>
      <name>ianywhere.connector.address</name>
      <value>simpleGroup</value>
    </prop>
    <prop>
      <client>simpleGroup</client>
      <name>ianywhere.connector.nativeConnection</name>
      <value>ianywhere.message.connector.group.GroupConnector</value>
    </prop>
    <prop>
      <client>simpleGroup</client>
      <name>ianywhere.connector.logLevel</name>
      <value>4</value>
    </prop>
    <prop>
      <client>simpleGroup</client>
      <name>ianywhere.qa.member.client1</name>
      <value>Y</value>
    </prop>
    <prop>
      <client>simpleGroup</client>
      <name>ianywhere.qa.member.client2\q11</name>
      <value>Y</value>
    </prop>
```



```
  </SetProperty>
  <OpenConnector>
    <client>simpleGroup</client>
  </OpenConnector>
</actions>
```

Adding and removing members in a destination alias

To add members to a destination alias, create a server management request that specifies the member in a property. The group must be restarted for the member setting to take effect.

The following example adds the member client3 and restarts the group simpleGroup:

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
    <SetProperty>
 <prop>
     <client>simpleGroup</client>
     <name>ianywhere.qa.member.client3</name>
     <value>Y</value>
 </prop>
    </SetProperty>
    <CloseConnector>
 <client>simpleGroup</client>
    </CloseConnector>
    <OpenConnector>
 <client>simpleGroup</client>
    </OpenConnector>
</actions>
```

To remove members from a destination alias, create a server management request that contains a property setting indicating that the member must be removed. The group must be restarted for the member removal setting to take effect.

The following example removes the member client3 and restarts the group simpleGroup:

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
    <SetProperty>
 <prop>
     <client>simpleGroup</client>
     <name>ianywhere.qa.member.client3</name>
 </prop>
    </SetProperty>
    <CloseConnector>
 <client>simpleGroup</client>
    </CloseConnector>
    <OpenConnector>
 <client>simpleGroup</client>
    </OpenConnector>
</actions>
```

QAnywhere monitoring

You can use a server management request to get information about a set of messages. The server compiles the information and sends it back to the client in a message. You can create a one-time message details

request or schedule your message details request to happen automatically. In addition, you can specify that your request should be persistent, so that the message is sent even if the server is restarted.

Message details requests

To write a server management request for message details, use the <MessageDetailsRequest> tag.

A message details request contains one or more <request> tags containing all the necessary information to register the request. Specifying more than one <request> tag is equivalent to sending multiple separate message details requests.

Use the optional <replyAddr> tag to specify the return address for each report generated as a result of the request. If this tag is omitted, the default return address of reports is the reply address of the originating message.

Use a <requestId> tag to specify a unique identifier for the request that is included in each report generated as a result of this request. Using different values for this field allows more than one request to be active at the same time. Using the same request ID allows the client to override or delete active requests.

Specify a <condition> tag to determine which messages should be included in the report.

You can also specify a list of details to determine what details of each message should be included in the report. You do this by including a set of empty detail element tags in the request.

You can use the <persistent> tag to specify that event details should be made persistent during any server downtime. This tag does not require any data and can be of the form <persistent/> or <persistent></persistent>.

You can use <schedule> to include all the necessary details needed to register a scheduled report.

<MessageDetailsRequest> subtags	**Description**
<request>	Groups information about a particular request. Specifying more than one <request> tag is equivalent to sending multiple separate server management requests for message information. See below.

Request tag

<Request> subtags	**Description**
<address>	Requests the address of each message.
<archived>	Requests whether the message is in the archive store.

<Request> subtags	Description
<condition>	Groups conditions for including a message in the report.
<contentSize>	Requests the content size of each message.
<expires>	Requests the expiration time of each message.
<kind>	Requests whether the message is text or binary.
<messageId>	Requests the message ID of each message.
<originator>	Requests the originator of each message.
<persistent>	Including this tag indicates that the results of the request should be made persistent in the server database (so that the report is sent even if the server is restarted).
<priority>	Requests the priority of each message.
<property>	Requests a list of all message properties and values for each message.
<statusTime>	Requests the status time of each message.
<replyAddr>	Specifies the return address for each report generated as a result of this request. If this tag is omitted, the default return address of reports is the reply address of the originating message.
<requestId>	This value is a unique identifier for the request and is included in each report generated as a result of this request. Using different values for this field allows more than one request to be active at the same time. Using the same request id allows the client to override or delete active requests.
<schedule>	Including this tag indicates that the report should be generated on a schedule. Subtags of <schedule> identify the schedule on which the report runs.
<status>	Requests the status of each message.
<transmissionStatus>	Requests the transmission status of each message.

See also

- "Server management requests" on page 147
- "Condition tag" on page 664
- "Server management request parent tags" on page 664
- "Server management request parent tags" on page 664

QAnywhere client monitoring

You can use a server management request to obtain a list of clients currently on the server. This list contains clients who are registered on the server, including remote clients, open connectors, and destination aliases.

To obtain a list of clients, use the <GetClientList> tag in your server management request. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
 <GetClientList/>   (or <GetClientList></GetClientList> )
</actions>
```

The response that is generated is sent to the reply address of the message containing the request. The response contains a list of <client> tags, each naming one client connected to the server. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<GetClientListResponse>
 <client>ianywhere.server</client>
 <client>ianywhere.connector.myConnector</client>
 <client>myClient</client>
</GetClientListResponse>
```

See also

- "Server management requests" on page 147

Property monitoring

You can use a server management request to see what properties are set for a client. The response lists only the properties that have been set for the client (not defaults).

To get a list of properties for a client, use the <GetProperties> tag in your server management request. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
 <GetProperties>
  <client>ianywhere.connector.myConnector</client>
 </GetProperties>
</actions>
```

The response that is generated is sent to the reply address of the message containing the request. The response contains the name of the client and a list of <prop> tags, each containing the details of one property. For example:

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



```
 <client>ianywhere.connector.myConnector</client>
 <prop>
  <name>ianywhere.connector.logLevel</name>
  <value>4</value>
 </prop>
 <prop>
  <name>ianywhere.connector.state</name>
  <value>2</value>
 </prop>
</GetPropertiesResponse>
```

See also

- “Server management requests” on page 147

Tutorial: Exploring TestMessage

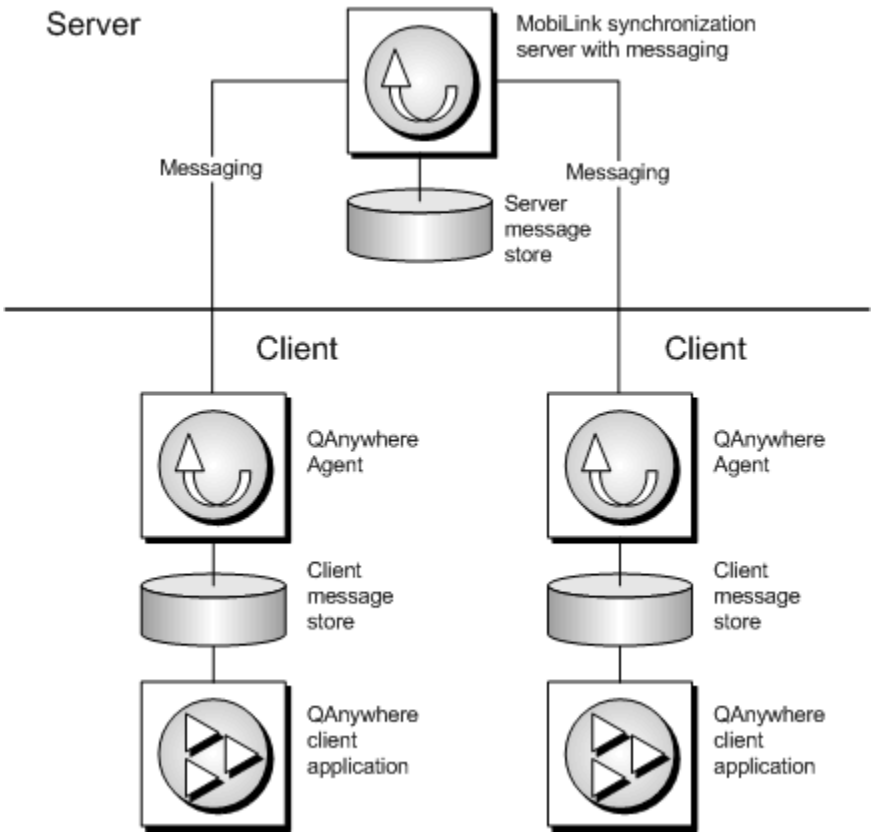
TestMessage is a sample QAnywhere client application. This application demonstrates how you can use QAnywhere to create your own messaging client applications. TestMessage provides a single client-to-client interface to send, receive, and display messages. Being human-readable, text messages provide a useful demonstration of QAnywhere messaging, but QAnywhere provides much more than text messaging. It is a general purpose application-to-application messaging system that provides message-based communication among many clients.

The tutorial is written for a Windows desktop system. While these platforms are convenient for demonstration purposes, you can also use the QAnywhere API to write applications that run on Windows Mobile devices. Source code is provided for Windows Mobile for C++, Visual Basic, C#, and Java. There is also a C# version written on the .NET Compact Framework.

Lesson 1: Starting MobiLink with messaging

Background

QAnywhere uses MobiLink synchronization to send and receive messages. All messages from one client to another are delivered through a central MobiLink server. The architecture of a typical system, with only two QAnywhere clients, is shown in the following diagram.



The server message store is a database configured for use as a MobiLink consolidated database. The TestMessage sample uses a SQL Anywhere consolidated database as its server message store.

The only tables needed in the server message store are the MobiLink system tables that are included in any supported database that is set up as a MobiLink consolidated database.

The system tables are maintained by MobiLink. A relational database provides a secure, high performance message store. It enables you to easily integrate messaging into an existing data management and synchronization system.

QAnywhere messaging is usually run over separate computers, but in this tutorial all components are running on a single computer. It is important to keep track of which activities are client activities and which are server activities.

In this lesson, you perform actions at the server.

Activity

The MobiLink server can be started with messaging by supplying the -m option, and specifying a connection string to the server message store. The TestMessage sample uses a QAnywhere sample database for the server message store. For the TestMessage sample, you can start the MobiLink server for messaging using the command line options, using a sample shortcut in your SQL Anywhere installation, or with the QAnywhere 12 plug-in to Sybase Central.

Start the messaging server

1. Click **Start** » **Programs** » **SQL Anywhere 12** » **QAnywhere** » **MobiLink QAnywhere Sample**.

   Alternatively, at a command prompt, run the following command:

   ```
   mlsrv12 -m -c "dsn=QAnywhere 12 Demo" -vcrs -zu+
   ```

   This command line uses the following mlsrv12 options:

Option	Description
-m	The -m option enables messaging. See "-m mlsrv12 option" [*MobiLink - Server Administration*].
-c	The -c option specifies the connection string to the server message store, in this case using the QAnywhere 12 Demo ODBC data source. See "-c mlsrv12 option" [*MobiLink - Server Administration*].
-vcrs	The -vcrs option provides verbose logging of server activities, which is useful during development. See "-v mlsrv12 option" [*MobiLink - Server Administration*].
-zu+	The -zu+ option automatically adds user names to the system; this is convenient for tutorial or development purposes but is not normally used in a production environment. See "-zu mlsrv12 option" [*MobiLink - Server Administration*].

2. Move the MobiLink server messages window to the center of your screen, which represents the server in this tutorial.

3. Proceed to "Lesson 2: Running the TestMessage application".

See also

- "Starting QAnywhere with MobiLink enabled" on page 29
- "-m mlsrv12 option" [*MobiLink - Server Administration*]
- "Quick start to QAnywhere" on page 10
- "Simple messaging scenario" on page 5

Lesson 2: Running the TestMessage application

Background

TestMessage is a simple application that uses QAnywhere to send and receive text messages. Text messaging is used in this tutorial because it provides a simple and accessible demonstration of messaging. QAnywhere is, however, not just a text messaging system; it provides general purpose application-to-application messaging.

In this lesson, you are performing activities at a client. Typically, clients run on separate computers from the server.

In this lesson, you start the client message store that is part of the TestMessage sample. In Lesson 3, you use this message store to send a message to another client message store.

Activity

Start the QAnywhere Agent with the TestMessage client message store

1. Click **Start** » **Programs** » **SQL Anywhere 12** » **QAnywhere** » **Tutorial Using SQL Anywhere** » **QAnywhere Agent For SQL Anywhere - saclient1**.

   The **QAnywhere Agent** connects to the first TestMessage sample client message store and manages message transmission to and from this message store.

2. Move the first **QAnywhere Agent** window to the right side of your screen.

   Note
   You must allow a few seconds for the first instance of the QAnywhere Agent to start before you proceed to the next step.

3. Click **Start** » **Programs** » **SQL Anywhere 12** » **QAnywhere** » **Tutorial Using SQL Anywhere** » **QAnywhere Agent For SQL Anywhere - saclient2**.

   A second **QAnywhere Agent** starts and connects to the second TestMessage sample client message store and manages message transmission to and from this message store.

4. Move the second **QAnywhere Agent** window to the left side of your screen.

Start TestMessage

1. Click **Start** » **Programs** » **SQL Anywhere 12** » **QAnywhere** » **Tutorial Using SQL Anywhere** » **TestMessage -- saclient1**.

2. Move the **saclient1 - TestMessage** window to the right side of your screen.

3. In the **saclient1 - TestMessage** window, click **Tools** » **Options**.

4. Verify **testmessage** appears in the **Queue Name Used To Listen For Incoming Messages** field. Click **Cancel**.

5. Click **Start** » **Programs** » **SQL Anywhere 12** » **QAnywhere** » **Tutorial Using SQL Anywhere** » **TestMessage -- saclient2**.

6. Move the **saclient2 - TestMessage** window to the left side of your screen.

7. In the **saclient2 - TestMessage** window, click **Tools** » **Options**.

8. Verify **testmessage** appears in the **Queue Name Used To Listen For Incoming Messages** field. Click **Cancel**.

Discussion

You can configure the way that the QAnywhere Agent monitors messages by setting a message transmission policy. This sample is designed to only work with the automatic or scheduled policy, and it starts the QAnywhere Agent using the automatic policy. The QAnywhere policies are:

- **scheduled** This policy setting instructs the QAnywhere Agent to transmit messages periodically. If you don't specify an interval, the default is 15 minutes.
- **automatic** This default policy setting causes the QAnywhere Agent to transmit messages whenever a message to or from the client message store is ready for delivery.
- **on demand** This policy setting causes the QAnywhere Agent to transmit messages only when instructed to by an application.
- **custom** In this mode, you provide a set of rules to specify more complicated transmission behavior.

QAnywhere messages are delivered to a QAnywhere address, which consists of a client message store ID and a queue name. The default ID is the computer name on which the QAnywhere Agent is running. Each message store requires its own QAnywhere Agent. Each application can listen to multiple queues, but each queue should be specific to a single application.

Proceed to "Lesson 3: Sending a message".

See also

- "Starting the QAnywhere agent" on page 44
- "Determining when message transmission should occur on the client" on page 46
- "qaagent utility" on page 672
- "QAnywhere transmission and delete rules" on page 729
- "How to write QAnywhere client applications" on page 49
- QAnywhere samples, which are installed to *%SQLANYSAMP12%\QAnywhere*.

Lesson 3: Sending a message

Background

In this lesson you send a message from the TestMessage saclient1 application to the TestMessage saclient2 application.

Activity

Send a message from TestMessage

Like other QAnywhere applications, TestMessage uses the QAnywhere API to manage messages. The QAnywhere API is supplied as a C++ API, a Java API, a Microsoft .NET API, and a SQL API.

1. In the **saclient1 - TestMessage** window, click **Message** » **New**.
2. In the **Destination ID** field, type **saclient2**.

3. In the **Subject** field, type the current time. Using the current time makes it easy to track individual messages.

4. In **Message** field, type **sample**.

5. Click **Send**.

6. In the **saclient2 - TestMessage** window, click the message. The content of the message appears in the bottom pane.

7. Proceed to "Lesson 4: Exploring the TestMessage client source code".

See also

- "QAnywhere message addresses" on page 57
- "Sending QAnywhere messages" on page 61
- "Message delete rules" on page 741

Lesson 4: Exploring the TestMessage client source code

Background

This section of the tutorial takes you on a brief tour of the source code behind the TestMessage client application.

A lot of the code implements the Windows interface, through which you can send, receive, and view the messages. This portion of the tutorial, however, focuses on the portions of the code given to QAnywhere.

You can find the TestMessage source code in *Samples\QAnywhere*.

Several versions of the TestMessage source code are provided. The following versions are provided for Windows platforms:

- A C++ version built using the Microsoft Foundation Classes is provided as *Samples\QAnywhere\Windows\MFC\TestMessage\TestMessage.sln*.
- A C# version built on the .NET Framework is provided as *Samples\QAnywhere\Windows\.NET\CS\TestMessage\TestMessage.sln*.
- A Java version is provided as *Samples\QAnywhere\Java\TestMessage\TestMessage.java*.

The following version is provided for .NET Compact Framework:

- A C# version built on the .NET Compact Framework is provided as *Samples\QAnywhere\Windows Mobile Classic\.NET\CS\TestMessage\TestMessage.sln*.

Required software

Visual Studio 2005 or later is required to open the solution files and build the .NET Framework projects and the .NET Compact Framework project.

Exploring the C# source

This section takes you through the C# source code. The two versions are structured in a very similar manner.

Rather than look at each line in the application, this lesson highlights particular lines that are useful for understanding QAnywhere applications. It uses the C# version to illustrate these lines.

1. Open the version of the TestMessage project that you are interested in.

   Double-click the solution file to open the project in Visual Studio. For example, *Samples\QAnywhere \Windows\.NET\CS\TestMessage\TestMessage.sln* is a solution file. There are several solution files for different environments.

2. Ensure that Solution Explorer is open.

   You can open the Solution Explorer from the **View** menu.

3. Inspect the **Source Files** folder.

   There are two files of particular importance. The *MessageList* file (*MessageList.cs*) receives messages and lets you view them. The *NewMessage* file (*NewMessage.cs*) allows you to construct and send messages.

4. From Solution Explorer, open the *MessageList* file.

5. Inspect the included namespaces.

   Every QAnywhere application requires the iAnywhere.QAnywhere.Client namespace. The assembly that defines this namespace is supplied as the DLL *iAnywhere.QAnywhere.Client.dll*. The files are in the following locations:

   - .NET Framework 2.0: *%SQLANY12%\Assembly\V2*
   - .NET Compact Framework 2.0: *%SQLANY12%\ce\Assembly\V2*

   For your own projects, you must include a reference to this DLL when compiling. The namespace is included using the following line at the top of each file:

   ```
   using iAnywhere.QAnywhere.Client;
   ```

6. Inspect the startReceiver method.

   This method performs initialization tasks that are common to QAnywhere applications:

   - Create a QAManager object.

     ```
     _qaManager =
     QAManagerFactory.Instance.CreateQAManager();
     ```

QAnywhere provides a QAManagerFactory object to create QAManager objects. The QAManager object handles QAnywhere messaging operations: in particular, receiving messages (getting messages from a queue) and sending messages (putting messages on a queue).

QAnywhere provides two types of manager: QAManager and QATransactionalManager. When using QATransactionalManager, all send and receive operations occur within a transaction, so that either all messages are sent (or received) or none are.

- Write a method to handle messages.

  The onMessage() method is called by QAnywhere to handle regular non-system messages. The message it receives is encoded as a QAMessage object. The QAMessage class and its children, QATextMessage and QABinaryMessage, provide properties and methods that hold all the information QAnywhere applications need about a message.

  ```
  private void onMessage( QAMessage msg ) {
     Invoke( new onMessageDelegate( onMessageReceived ),
        new Object [] { msg } );
  }
  ```

  This code uses the Invoke method of the Form to cause the event to be processed on the thread that runs the underlying window so that the user interface can be updated to display the message. This is also the thread that created the QAManager. With some exceptions, the QAManager can only be accessed from the thread that created it.

- Declare a MessageListener, as defined in the MessageList_Load method.

  ```
  _receiveListener = new
      QAManager.MessageListener( onMessage );
  ```

  The OnMessage() method is called whenever a message is received by the QAnywhere Agent and placed in the queue that the application listens to.

> **Note**
> Message listeners are different from the MobiLink Listener component described in "Scenario for messaging with push notifications".
>
> The MobiLink Listener component receives notifications, while message listener objects retrieve messages from the queue.

When you set a message listener for the queue, the QAnywhere Manager passes messages that arrive on that queue to that listener. Only one listener can be set for a given queue. Setting with a null listener clears out any listener for that queue.

The MessageListener implementation receives messages asynchronously. You can also receive messages synchronously; that is, the application explicitly goes and looks for messages on the queue, perhaps in response to a user action such as clicking a Refresh button, rather than being notified when messages appear.

Other initialization tasks include:

- Open and start the QAManager object.


```
_qaManager.Open(
        AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT );
_qaManager.Start();
```

The AcknowledgementMode enumeration constants determine how the receipt of messages is acknowledged to the sender. The EXPLICIT_ACKNOWLEDGEMENT constant indicates that messages are not acknowledged until a call to one of the QAManager acknowledge methods is made.

- Load any messages that are waiting in the queue.

  ```
  loadMessages();
  ```

- Assign a listener to a queue for future messages.

  The listener was declared in the MessageList_Load() method.

  ```
  _qaManager.SetMessageListener(
    _options.ReceiveQueueName,
    _receiveListener );
  ```

  The Options ReceiveQueueName property contains the string **testmessage**, which is the TestMessage queue as set in the **TestMessage Options** window.

7. Inspect the addMessage() method in the same file.

   This method is called whenever the application receives a message. It gets properties of the message such as its reply-to address, preferred name, and the time it was sent (Timestamp), and displays the information in the TestMessage user interface. The following lines cast the incoming message into a QATextMessage object and get the reply-to address of the message:

   ```
   text_msg = ( QATextMessage )msg;
   from = text_msg.ReplyToAddress;
   ```

   This completes a brief look at some of the major tasks in the *MessageList* file.

8. From Solution Explorer, open the *NewMessage* file.

9. Inspect the sendMessage() method.

   This method takes the information entered in the **New Message** window and constructs a QATextMessage object. The QAManager then puts the message in the queue to be sent.

   Here are the lines that create a QATextMessage object and set its ReplyToAddress property:

   ```
   qa_manager = MessageList.GetQAManager();
   msg = qa_manager.CreateTextMessage();
   msg.ReplyToAddress = MessageList.getOptions().ReceiveQueueName;
   ```

   Here are the lines that put the message in the queue to be sent. The variable `dest` is the destination address, supplied as an argument to the function.

   ```
   qa_manager.PutMessage( dest, msg );
   ```

10. Proceed to “Cleaning up”.

See also

- "QAnywhere C++ API reference for clients" on page 354
- "QAnywhere .NET API reference for clients" on page 181
- "How to write QAnywhere client applications" on page 49
- The TestMessage sample, which is installed to *%SQLANYSAMP12%\QAnywhere*.

Cleaning up

Shut down all instances of TestMessage, the QAnywhere Agent, and the MobiLink server.

QAnywhere reference

This section provides reference documentation of the QAnywhere APIs.

QAnywhere .NET API reference for clients

Namespace (for regular clients)

```
iAnywhere.QAnywhere.Client
```

Namespace (for standalone clients)

```
iAnywhere.QAnywhere.StandAloneClient
```

MessageProperties class

Provides fields storing standard message property names.

Visual Basic syntax

```
Public Class MessageProperties
```

C# syntax

```
public class MessageProperties
```

Members

All members of MessageProperties class, including all inherited members.

Name	Description
ADAPTER field	For system queue messages, the network adapter that is being used to connect to the QAnywhere server.
ADAPTERS field	This property name refers to a delimited list of network adapters that can be used to connect to the QAnywhere server.
DELIVERY_COUNT field	This property name refers to the number of attempts that have been made so far to deliver the message.
IP field	For system queue messages, the IP address of the network adapter that is being used to connect to the QAnywhere server.
MAC field	For system queue messages, the MAC address of the network adapter that is being used to connect to the QAnywhere server.

Name	Description
MSG_TYPE field	This property name refers to MessageType values associated with a QAnywhere message.
NETWORK_STATUS field	This property name refers to the state of the network connection.
ORIGINATOR field	This property name refers to the message store ID of the originator of the message.
RAS field	For system queue messages, the RAS entry name that is being used to connect to the QAnywhere server.
RASNAMES field	For system queue messages, a delimited list of RAS entry names that can be used to connect to the QAnywhere server.
STATUS field	This property name refers to the current status of the message.
STATUS_TIME field	This property name refers to the time at which the message became its current status.
TRANSMISSION_STATUS field	This property name refers to the current transmission status of the message.

Remarks

The MessageProperties class provides standard message property names. You can pass MessageProperties fields to QAMessage methods used to get and set message properties.

For more information, see “QAnywhere messages” on page 13.

See also

- “QAMessage interface [QAnywhere .NET]” on page 272
- “QAMessage.GetIntProperty method [QAnywhere .NET]” on page 277
- “QAMessage.GetStringProperty method [QAnywhere .NET]” on page 281

Example

Assume you have the following QAMessage instance:

```
QAMessage msg = mgr.createTextMessage();
```

The following example gets the value corresponding to MessageProperties.MSG_TYPE using the QAMessage.GetIntProperty method. The MessageType enumeration maps the integer result to an appropriate message type.

```
msg_type = (MessageType)t_msg.GetIntProperty(
    MessageProperties.MSG_TYPE
);
```


The following example shows the onSystemMessage method which is used to handle QAnywhere system messages. The message type is evaluated using MessageProperties.MSG_TYPE variable and the QAMessage.GetIntProperty method. A delimited list of RAS entry names is obtained using MessageProperties.RASNAMES and the QAMessage.GetStringProperty method.

```
private void onSystemMessage(QAMessage msg) {
    QATextMessage    t_msg;
    MessageType        msg_type;
    String             network_adapters;
    String             network_names;
    String             network_info;

    t_msg = (QATextMessage)msg;
    if( t_msg != null ) {
          // Evaluate message type.
          msg_type =
(MessageType)t_msg.GetIntProperty( MessageProperties.MSG_TYPE );

        if( msg_type == MessageType.NETWORK_STATUS_NOTIFICATION ) {
            // Handle network status notification.
            network_info = "";
            network_adapters =
t_msg.GetStringProperty( MessageProperties.ADAPTERS );
            if( network_adapters != null && network_adapters.Length > 0) {
                network_info +=
String.Format( _resources.GetString( "NetworkAdapter" ), network_adapters );
            }

            network_names =
t_msg.GetStringProperty( MessageProperties.RASNAMES );
            //...
          }
    }
}
```

ADAPTER field

For system queue messages, the network adapter that is being used to connect to the QAnywhere server.

Visual Basic syntax

```
Public Const ADAPTER As String
```

C# syntax

```
public const string ADAPTER;
```

Remarks

The value of this field is ias_Network.Adapter.

For more information, see “Predefined client message store properties” on page 717.

You can pass MessageProperties.ADAPTER in the QAMessage.GetStringProperty method to access the associated property.

See also

- "MessageProperties class [QAnywhere .NET]" on page 181
- "QAMessage.GetStringProperty method [QAnywhere .NET]" on page 281

ADAPTERS field

This property name refers to a delimited list of network adapters that can be used to connect to the QAnywhere server.

Visual Basic syntax

```
Public Const ADAPTERS As String
```

C# syntax

```
public const string ADAPTERS;
```

Remarks

It is used for system queue messages.

You can pass MessageProperties.ADAPTERS in the QAMessage.GetStringProperty method to access the associated property.

For more information, see "Predefined client message store properties" on page 717.

See also

- "MessageProperties class [QAnywhere .NET]" on page 181
- "QAMessage.GetStringProperty method [QAnywhere .NET]" on page 281

DELIVERY_COUNT field

This property name refers to the number of attempts that have been made so far to deliver the message.

Visual Basic syntax

```
Public Const DELIVERY_COUNT As String
```

C# syntax

```
public const string DELIVERY_COUNT;
```

Remarks

The value of this field is ias_DeliveryCount.

You can pass MessageProperties.DELIVERY_COUNT in the QAMessage.GetIntProperty method to access the associated property.

See also

- "MessageProperties class [QAnywhere .NET]" on page 181
- "QAMessage.GetIntProperty method [QAnywhere .NET]" on page 277

IP field

For system queue messages, the IP address of the network adapter that is being used to connect to the QAnywhere server.

Visual Basic syntax

```
Public Const IP As String
```

C# syntax

```
public const string IP;
```

Remarks

The value of this field is ias_Network.IP.

For more information, see "Predefined client message store properties" on page 717.

You can pass MessageProperties.IP in the QAMessage.GetStringProperty method to access the associated property.

See also

- "MessageProperties class [QAnywhere .NET]" on page 181
- "QAMessage.GetStringProperty method [QAnywhere .NET]" on page 281

MAC field

For system queue messages, the MAC address of the network adapter that is being used to connect to the QAnywhere server.

Visual Basic syntax

```
Public Const MAC As String
```

C# syntax

```
public const string MAC;
```

Remarks

The value of this field is ias_Network.MAC.

For more information, see "Predefined client message store properties" on page 717.

You can pass MessageProperties.MAC in the QAMessage.GetStringProperty method to access the associated property.

See also

- "MessageProperties class [QAnywhere .NET]" on page 181
- "QAMessage.GetStringProperty method [QAnywhere .NET]" on page 281

MSG_TYPE field

This property name refers to MessageType values associated with a QAnywhere message.

Visual Basic syntax

```
Public Const MSG_TYPE As String
```

C# syntax

```
public const string MSG_TYPE;
```

Remarks

The value of this field is ias_MessageType.

You can pass MessageProperties.MSG_TYPE in the QAMessage.GetIntProperty method to access the associated property.

See also

- "MessageProperties class [QAnywhere .NET]" on page 181
- "MessageType enumeration [QAnywhere .NET]" on page 303
- "QAMessage.GetIntProperty method [QAnywhere .NET]" on page 277
- "QAMessage.GetStringProperty method [QAnywhere .NET]" on page 281

NETWORK_STATUS field

This property name refers to the state of the network connection.

Visual Basic syntax

```
Public Const NETWORK_STATUS As String
```

C# syntax

```
public const string NETWORK_STATUS;
```

Remarks

The value is 1 if the network is accessible and 0 otherwise.

The network status is used for system queue messages (for example, network status changes).

For more information, see "Predefined client message store properties" on page 717.

You can pass MessageProperties.NETWORK_STATUS in the QAMessage.GetIntProperty method to access the associated property.

See also

- "MessageProperties class [QAnywhere .NET]" on page 181
- "QAMessage.GetIntProperty method [QAnywhere .NET]" on page 277

ORIGINATOR field

This property name refers to the message store ID of the originator of the message.

Visual Basic syntax

```
Public Const ORIGINATOR As String
```

C# syntax

```
public const string ORIGINATOR;
```

Remarks

The value of this field is ias_Originator.

You can pass MessageProperties.ORIGINATOR in the QAMessage.GetStringProperty method to access the associated property.

See also

- "MessageProperties class [QAnywhere .NET]" on page 181
- "QAMessage.GetStringProperty method [QAnywhere .NET]" on page 281

RAS field

For system queue messages, the RAS entry name that is being used to connect to the QAnywhere server.

Visual Basic syntax

```
Public Const RAS As String
```

C# syntax

```
public const string RAS;
```

Remarks

The value of this field is ias_Network.RAS.

For more information, see "Predefined client message store properties" on page 717.

You can pass MessageProperties.RAS in the QAMessage.GetStringProperty method to access the associated property.

See also

- "MessageProperties class [QAnywhere .NET]" on page 181
- "QAMessage.GetStringProperty method [QAnywhere .NET]" on page 281

RASNAMES field

For system queue messages, a delimited list of RAS entry names that can be used to connect to the QAnywhere server.

Visual Basic syntax

```
Public Const RASNAMES As String
```

C# syntax

```
public const string RASNAMES;
```

Remarks

The value of this field is ias_RASNames.

For more information, see "Predefined client message store properties" on page 717.

You can pass MessageProperties.RASNAMES in the QAMessage.GetStringProperty method to access the associated property.

See also

- "MessageProperties class [QAnywhere .NET]" on page 181
- "QAMessage.GetStringProperty method [QAnywhere .NET]" on page 281

STATUS field

This property name refers to the current status of the message.

Visual Basic syntax

```
Public Const STATUS As String
```

C# syntax

```
public const string STATUS;
```

Remarks

For a list of property values, see the StatusCodes.

The value of this field is ias_Status.

You can pass MessageProperties.STATUS in the QAMessage.GetIntProperty method to access the associated property.

See also

- "MessageProperties class [QAnywhere .NET]" on page 181
- "QAMessage.GetIntProperty method [QAnywhere .NET]" on page 277

STATUS_TIME field

This property name refers to the time at which the message became its current status.

Visual Basic syntax

```
Public Const STATUS_TIME As String
```

C# syntax

```
public const string STATUS_TIME;
```

Remarks

It is a local time. When STATUS_TIME is passed to QAMessage.GetProperty, it returns a DateTime object. The value of this field is ias_StatusTime.

See also

- "QAMessage.GetProperty method [QAnywhere .NET]" on page 278
- "MessageProperties class [QAnywhere .NET]" on page 181
- "QAMessage.GetProperty method [QAnywhere .NET]" on page 278

TRANSMISSION_STATUS field

This property name refers to the current transmission status of the message.

Visual Basic syntax

```
Public Const TRANSMISSION_STATUS As String
```

C# syntax

```
public const string TRANSMISSION_STATUS;
```

Remarks

For a list of property values, see the StatusCodes.

The value of this field is ias_TransmissionStatus.

You can pass MessageProperties.TRANSMISSION_STATUS in the QAMessage.GetIntProperty method to access the associated property.

See also

- "MessageProperties class [QAnywhere .NET]" on page 181
- "QAMessage.GetIntProperty method [QAnywhere .NET]" on page 277

MessageStoreProperties class

This class defines constant values for useful message store property names.

Visual Basic syntax

```
Public Class MessageStoreProperties
```

C# syntax

```
public class MessageStoreProperties
```

Members

All members of MessageStoreProperties class, including all inherited members.

Name	Description
MAX_DELIVERY_ATTEMPTS field	This property name refers to the maximum number of times that a message can be received, without explicit acknowledgement, before its status is set to StatusCodes.UNRECEIVABLE.

Remarks

The MessageStoreProperties class provides standard message property names. You can pass MessageProperties fields to QAManagerBase methods used to get and set predefined or custom message store properties.

For more information, see "Client message store properties" on page 26.

MAX_DELIVERY_ATTEMPTS field

This property name refers to the maximum number of times that a message can be received, without explicit acknowledgement, before its status is set to StatusCodes.UNRECEIVABLE.

Visual Basic syntax

```
Public Const MAX_DELIVERY_ATTEMPTS As String
```

C# syntax

```
public const string MAX_DELIVERY_ATTEMPTS;
```

Remarks

The value of this field is ias_MaxDeliveryAttempts.

iAnywhere.QAnywhere.Client.StatusCodes MessageStoreProperties

QABinaryMessage interface

An QABinaryMessage object is used to send a message containing a stream of uninterpreted bytes.

Visual Basic syntax

```
Public Interface QABinaryMessage Inherits QAMessage
```

C# syntax

```
public interface QABinaryMessage : QAMessage
```

Base classes

- "QAMessage interface [QAnywhere .NET]" on page 272

Members

All members of QABinaryMessage interface, including all inherited members.

Name	Description
ClearBody method	Clears the body of the message.
ClearProperties method	Clears all the properties of the message.
GetBooleanProperty method	Gets a boolean message property.
GetByteProperty method	Gets a byte message property.
GetDoubleProperty method	Gets a double message property.
GetFloatProperty method	Gets a float message property.
GetIntProperty method	Gets an int message property.
GetLongProperty method	Gets a long message property.
GetProperty method	Gets a message property.
GetPropertyNames method	Gets an enumerator over the property names of the message.
GetPropertyType method	Returns the property type of the given property.
GetSbyteProperty method	Gets a signed byte message property.
GetShortProperty method	Gets a short message property.
GetStringProperty method	Gets a string message property.
PropertyExists method	Indicates whether the given property has been set for this message.

Name	Description
ReadBinary method	Reads up to length number of bytes starting from the unread portion of a QABinaryMessage instance body and stores them into the **dest** array.
ReadBoolean method	Reads a boolean value starting from the unread portion of the QABinaryMessage instance's message body.
ReadChar method	Reads a char value starting from the unread portion of a QABinaryMessage message body.
ReadDouble method	Reads a double value starting from the unread portion of a QABinaryMessage message body.
ReadFloat method	Reads a float value starting from the unread portion of a QABinaryMessage message body.
ReadInt method	Reads an integer value starting from the unread portion of a QABinaryMessage message body.
ReadLong method	Reads a long value starting from the unread portion of a QABinaryMessage message body.
ReadSbyte method	Reads a signed byte value starting from the unread portion of a QABinaryMessage message body.
ReadShort method	Reads a short value starting from the unread portion of a QABinaryMessage message body.
ReadString method	Reads a string value starting from the unread portion of a QABinaryMessage message body.
Reset method	Resets a message so that the reading of values starts from the beginning of the message body.
SetBooleanProperty method	Sets a boolean property.
SetByteProperty method	Sets a byte property.
SetDoubleProperty method	Sets a double property.
SetFloatProperty method	Sets a float property.
SetIntProperty method	Sets an int property.
SetLongProperty method	Sets a long property.
SetProperty method	Sets a property.

Name	Description
SetSbyteProperty method	Sets a signed byte property.
SetShortProperty method	Sets a short property.
SetStringProperty method	Sets a string property.
WriteBinary method	Appends length bytes from a byte array starting at the given offset to the QABinaryMessage instance's message body.
WriteBoolean method	Appends a boolean value to the QABinaryMessage instance's message body.
WriteChar method	Appends a char value to the QABinaryMessage instance's message body.
WriteDouble method	Appends a double value to the QABinaryMessage instance's message body.
WriteFloat method	Appends a float value to the QABinaryMessage instance's message body.
WriteInt method	Appends an integer value to the QABinaryMessage instance's message body.
WriteLong method	Appends a long value to the QABinaryMessage instance's message body.
WriteSbyte method	Appends a signed byte value to the QABinaryMessage instance's message body.
WriteShort method	Appends a short value to the QABinaryMessage instance's message body.
WriteString method	Appends a string value to the QABinaryMessage instance's message body.
Address property	The destination address for the QAMessage instance.
BodyLength property	Returns the size of the message body in bytes.
Expiration property	Gets the message's expiration value.
InReplyToID property	The message id of the message for which this message is a reply.
MessageID property	The globally unique message id of the message.
Priority property	The priority of the message (ranging from 0 to 9).

Name	Description
Redelivered property	Indicates whether the message has been previously received but not acknowledged.
ReplyToAddress property	The reply to address of this message.
Timestamp property	The message timestamp.

Remarks

It inherits from the QAMessage class and adds a bytes message body. QABinaryMessage provides a variety of functions to read from and write to the bytes message body.

When the message is first created, the body of the message is in write-only mode. After a message has been sent, the client that sent it can retain and modify it without affecting the message that has been sent. The same message object can be sent multiple times.

When a message is received, the provider has called QABinaryMessage.Reset() so that the message body is in read-only mode and reading of values starts from the beginning of the message body.

Example

The following example uses the QABinaryMessage writeString function to write the string "Q" followed by the string "Anywhere" to a QABinaryMessage instances message body.

```
// create a binary message instance
QABinaryMessage binary_message;
binary_message = qa_manager.CreateBinaryMessage();

// set optional message properties ...
binary_message.ReplyToAddress = "my-queue-name";

// write to the message body
binary_message->WriteString("Q");
binary_messge->WriteString("Anywhere");

// put the message in the local database, ready for sending
if(!qa_manager->putMessage( "store-id\\queue-name", msg )) {
        handleError();
    }
```

The message is sent by the QAnywhere Agent. On the receiving end, the first QABinaryMessage.ReadString invocation returns "Q", and the next QABinaryMessage.ReadString invocation returns "Anywhere".

For more information, see "Determining when message transmission should occur on the client" on page 46 and "How to write QAnywhere client applications" on page 49.

ReadBinary method

Reads up to length number of bytes starting from the unread portion of a QABinaryMessage instance body and stores them into the dest array.

Overload list

Name	Description
ReadBinary(byte[]) method	Reads some number of bytes starting from the unread portion of a QABinaryMessage instance body and stores them into the **dest** array.
ReadBinary(byte[], int) method	Reads up to length number of bytes starting from the unread portion of a QABinaryMessage instance body and stores them into the **dest** array.
ReadBinary(byte[], int, int) method	Reads up to length number of bytes starting from the unread portion of a QABinaryMessage instance body and stores them into the **dest** array starting at dest[offset].

ReadBinary(byte[]) method

Reads some number of bytes starting from the unread portion of a QABinaryMessage instance body and stores them into the **dest** array.

Visual Basic syntax

```
Public Function ReadBinary(ByVal dest As Byte()) As Integer
```

C# syntax

```
public int ReadBinary(byte[] dest)
```

Parameters

- **dest** The byte array that will contain the read bytes.

Returns

The number of bytes read from the message body, or -1 if there are no more bytes available.

Exceptions

- **QAException class** Thrown if there was an error reading bytes from the message.

Remarks

The ReadBinary(dest) method has the same effect as: ReadBinary(dest,0,dest.Length)

See also

- "QABinaryMessage interface [QAnywhere .NET]" on page 191
- "QABinaryMessage.WriteBinary method [QAnywhere .NET]" on page 202

ReadBinary(byte[], int) method

Reads up to length number of bytes starting from the unread portion of a QABinaryMessage instance body and stores them into the **dest** array.

Visual Basic syntax

```
Public Function ReadBinary(
     ByVal dest As Byte(),
    ByVal length As Integer
) As Integer
```

C# syntax

```
public int ReadBinary(byte[] dest, int length)
```

Parameters

- **dest** The byte array that will contain the read bytes.
- **length** The maximum number of bytes to read.

Returns

The number of bytes read from the message body, or -1 if there are no more bytes available.

Exceptions

- **QAException class** Thrown if there was an error reading bytes

Remarks

The ReadBinary(dest,len) method has the same effect as: ReadBinary(dest,0,len)

See also

- "QABinaryMessage interface [QAnywhere .NET]" on page 191
- "QABinaryMessage.WriteBinary method [QAnywhere .NET]" on page 202

ReadBinary(byte[], int, int) method

Reads up to length number of bytes starting from the unread portion of a QABinaryMessage instance body and stores them into the **dest** array starting at dest[offset].

Visual Basic syntax

```
Public Function ReadBinary(
     ByVal dest As Byte(),
    ByVal offset As Integer,
    ByVal length As Integer
) As Integer
```

C# syntax

```
public int ReadBinary(byte[] dest, int offset, int length)
```

Parameters

- **dest** The byte array that will contain the read bytes.
- **offset** The start offset of the destination array.

- **length** The maximum number of bytes to read.

Returns

The number of bytes read from the message body, or -1 if there are no more bytes available

Exceptions

- **QAException class** Thrown if there was a conversion error reading the value or if there is no more input.

Remarks

If dest is null, an ArgumentNullException is thrown. If offset is negative, or length is negative, or offset +length is greater than the length of dest, then an ArgumentOutOfRangeException is thrown.

See also

- "QABinaryMessage interface [QAnywhere .NET]" on page 191
- "QABinaryMessage.WriteBinary method [QAnywhere .NET]" on page 202

ReadBoolean method

Reads a boolean value starting from the unread portion of the QABinaryMessage instance's message body.

Visual Basic syntax

```
Public Function ReadBoolean() As Boolean
```

C# syntax

```
public bool ReadBoolean()
```

Returns

The boolean value read from the message body.

Exceptions

- **QAException class** Thrown if there was a conversion error reading the value or if there is no more input.

See also

- "QABinaryMessage interface [QAnywhere .NET]" on page 191
- "QABinaryMessage.WriteBoolean method [QAnywhere .NET]" on page 204

ReadChar method

Reads a char value starting from the unread portion of a QABinaryMessage message body.

Visual Basic syntax

```
Public Function ReadChar() As Char
```

C# syntax

```
public char ReadChar()
```

Returns

The character value read from the message body.

Exceptions

- **QAException class** if there was a conversion error reading the value or if there is no more input.

See also

- "QABinaryMessage interface [QAnywhere .NET]" on page 191
- "QABinaryMessage.WriteChar method [QAnywhere .NET]" on page 204

ReadDouble method

Reads a double value starting from the unread portion of a QABinaryMessage message body.

Visual Basic syntax

```
Public Function ReadDouble() As Double
```

C# syntax

```
public double ReadDouble()
```

Returns

The double value read from the message body.

Exceptions

- **QAException class** Thrown if there was a conversion error reading the value or if there is no more input.

See also

- "QABinaryMessage interface [QAnywhere .NET]" on page 191
- "QABinaryMessage.WriteDouble method [QAnywhere .NET]" on page 205

ReadFloat method

Reads a float value starting from the unread portion of a QABinaryMessage message body.

Visual Basic syntax

```
Public Function ReadFloat() As Single
```


C# syntax

```
public float ReadFloat()
```

Returns

The float value read from the message body.

Exceptions

- **QAException class** Thrown if there was a conversion error reading the value or if there is no more input.

See also

- "QABinaryMessage interface [QAnywhere .NET]" on page 191
- "QABinaryMessage.WriteFloat method [QAnywhere .NET]" on page 205

ReadInt method

Reads an integer value starting from the unread portion of a QABinaryMessage message body.

Visual Basic syntax

```
Public Function ReadInt() As Integer
```

C# syntax

```
public int ReadInt()
```

Returns

The int value read from the message body.

Exceptions

- **QAException class** Thrown if there was a conversion error reading the value or if there is no more input.

See also

- "QABinaryMessage interface [QAnywhere .NET]" on page 191
- "QABinaryMessage.WriteInt method [QAnywhere .NET]" on page 206

ReadLong method

Reads a long value starting from the unread portion of a QABinaryMessage message body.

Visual Basic syntax

```
Public Function ReadLong() As Long
```

C# syntax

```
public long ReadLong()
```

Returns

The long value read from the message body.

Exceptions

- **QAException class** Thrown if there was a conversion error reading the value or if there is no more input.

See also

- "QABinaryMessage interface [QAnywhere .NET]" on page 191
- "QABinaryMessage.WriteLong method [QAnywhere .NET]" on page 206

ReadSbyte method

Reads a signed byte value starting from the unread portion of a QABinaryMessage message body.

Visual Basic syntax

```
Public Function ReadSbyte() As SByte
```

C# syntax

```
public sbyte ReadSbyte()
```

Returns

The signed byte value read from the message body.

Exceptions

- **QAException class** Thrown if there was a conversion error reading the value or if there is no more input.

See also

- "QABinaryMessage interface [QAnywhere .NET]" on page 191
- "QABinaryMessage.WriteSbyte method [QAnywhere .NET]" on page 207

ReadShort method

Reads a short value starting from the unread portion of a QABinaryMessage message body.

Visual Basic syntax

```
Public Function ReadShort() As Short
```


C# syntax

```
public short ReadShort()
```

Returns

The short value read from the message body.

Exceptions

- **QAException class** Thrown if there was a conversion error reading the value or if there is no more input.

See also

- "QABinaryMessage interface [QAnywhere .NET]" on page 191
- "QABinaryMessage.WriteShort method [QAnywhere .NET]" on page 207

ReadString method

Reads a string value starting from the unread portion of a QABinaryMessage message body.

Visual Basic syntax

```
Public Function ReadString() As String
```

C# syntax

```
public string ReadString()
```

Returns

The string value read from the message body.

Exceptions

- **QAException class** Thrown if there was a conversion error reading the value or if there is no more input.

See also

- "QABinaryMessage interface [QAnywhere .NET]" on page 191
- "QABinaryMessage.WriteString method [QAnywhere .NET]" on page 208

Reset method

Resets a message so that the reading of values starts from the beginning of the message body.

Visual Basic syntax

```
Public Sub Reset()
```

C# syntax

```
public void Reset()
```

Remarks

The Reset method also puts the QABinaryMessage message body in read-only mode.

See also

- "QABinaryMessage interface [QAnywhere .NET]" on page 191

WriteBinary method

Appends length bytes from a byte array starting at the given offset to the QABinaryMessage instance's message body.

Overload list

Name	Description
WriteBinary(byte[]) method	Appends a byte array value to the QABinaryMessage instance's message body.
WriteBinary(byte[], int) method	Appends length bytes from a byte array to the QABinaryMessage instance's message body.
WriteBinary(byte[], int, int) method	Appends length bytes from a byte array starting at the given offset to the QABinaryMessage instance's message body.

WriteBinary(byte[]) method

Appends a byte array value to the QABinaryMessage instance's message body.

Visual Basic syntax

```
Public Sub WriteBinary(ByVal val As Byte())
```

C# syntax

```
public void WriteBinary(byte[] val)
```

Parameters

- **val** The byte array value to write to the message body.

Exceptions

- **QAException class** Thrown if there was a problem appending the byte array to the message body

Remarks

The WriteBinary(val) method has the same effect as: WriteBinary(val,0,val.Length)

See also

- "QABinaryMessage interface [QAnywhere .NET]" on page 191
- "QABinaryMessage.ReadBinary method [QAnywhere .NET]" on page 194

WriteBinary(byte[], int) method

Appends length bytes from a byte array to the QABinaryMessage instance's message body.

Visual Basic syntax

```
Public Sub WriteBinary(ByVal val As Byte(), ByVal length As Integer)
```

C# syntax

```
public void WriteBinary(byte[] val, int length)
```

Parameters

- **val** The byte array value to write to the message body.
- **length** The number of bytes to write.

Exceptions

- **QAException class** Thrown if there was a problem appending the byte array to the message body

Remarks

The WriteBinary(val,len) method has the same effect as: WriteBinary(val,0,len)

See also

- "QABinaryMessage interface [QAnywhere .NET]" on page 191
- "QABinaryMessage.ReadBinary method [QAnywhere .NET]" on page 194

WriteBinary(byte[], int, int) method

Appends length bytes from a byte array starting at the given offset to the QABinaryMessage instance's message body.

Visual Basic syntax

```
Public Sub WriteBinary(
     ByVal val As Byte(),
    ByVal offset As Integer,
    ByVal length As Integer
)
```

C# syntax

```
public void WriteBinary(byte[] val, int offset, int length)
```

Parameters

- **val** The byte array value to write to the message body.

- **offset** The offset within the byte array to begin writing.
- **length** The number of bytes to write.

Exceptions

- **QAException class** Thrown if there was a problem appending the byte array to the message body

Remarks

If val is null, a ArgumentNullException is thrown. If offset is negative, or length is negative, or length +offset is greater than the length of val then an ArgumentOutOfRangeException is thrown.

See also

- "QABinaryMessage interface [QAnywhere .NET]" on page 191
- "QABinaryMessage.ReadBinary method [QAnywhere .NET]" on page 194

WriteBoolean method

Appends a boolean value to the QABinaryMessage instance's message body.

Visual Basic syntax

```
Public Sub WriteBoolean(ByVal val As Boolean)
```

C# syntax

```
public void WriteBoolean(bool val)
```

Parameters

- **val** The boolean value to write to the message body.

Remarks

The boolean is represented as a one-byte value. True is represented as 1; false is represented as 0.

See also

- "QABinaryMessage interface [QAnywhere .NET]" on page 191
- "QABinaryMessage.ReadBoolean method [QAnywhere .NET]" on page 197

WriteChar method

Appends a char value to the QABinaryMessage instance's message body.

Visual Basic syntax

```
Public Sub WriteChar(ByVal val As Char)
```


C# syntax

```
public void WriteChar(char val)
```

Parameters

- **val** The char value to write to the message body.

Remarks

The char is represented as a two byte value and the high order byte is appended first.

See also

- "QABinaryMessage interface [QAnywhere .NET]" on page 191
- "QABinaryMessage.ReadChar method [QAnywhere .NET]" on page 197

WriteDouble method

Appends a double value to the QABinaryMessage instance's message body.

Visual Basic syntax

```
Public Sub WriteDouble(ByVal val As Double)
```

C# syntax

```
public void WriteDouble(double val)
```

Parameters

- **val** The double value to write to the message body.

Remarks

The double is converted to a representative 8-byte long and higher order bytes are appended first.

See also

- "QABinaryMessage interface [QAnywhere .NET]" on page 191
- "QABinaryMessage.ReadDouble method [QAnywhere .NET]" on page 198

WriteFloat method

Appends a float value to the QABinaryMessage instance's message body.

Visual Basic syntax

```
Public Sub WriteFloat(ByVal val As Single)
```

C# syntax

```
public void WriteFloat(float val)
```

Parameters

- **val** The float value to write to the message body.

Remarks

The float parameter is converted to a representative 4-byte integer and the higher order bytes are appended first.

See also

- "QABinaryMessage interface [QAnywhere .NET]" on page 191
- "QABinaryMessage.ReadFloat method [QAnywhere .NET]" on page 198

WriteInt method

Appends an integer value to the QABinaryMessage instance's message body.

Visual Basic syntax

```
Public Sub WriteInt(ByVal val As Integer)
```

C# syntax

```
public void WriteInt(int val)
```

Parameters

- **val** The int value to write to the message body.

Remarks

The integer parameter is represented as a 4 byte value and higher order bytes are appended first.

See also

- "QABinaryMessage interface [QAnywhere .NET]" on page 191
- "QABinaryMessage.ReadInt method [QAnywhere .NET]" on page 199

WriteLong method

Appends a long value to the QABinaryMessage instance's message body.

Visual Basic syntax

```
Public Sub WriteLong(ByVal val As Long)
```

C# syntax

```
public void WriteLong(long val)
```

Parameters

- **val** The long value to write to the message body.

Remarks

The long parameter is represented using an 8-byte value and higher order bytes are appended first.

See also

- "QABinaryMessage interface [QAnywhere .NET]" on page 191
- "QABinaryMessage.ReadLong method [QAnywhere .NET]" on page 199

WriteSbyte method

Appends a signed byte value to the QABinaryMessage instance's message body.

Visual Basic syntax

```
Public Sub WriteSbyte(ByVal val As SByte)
```

C# syntax

```
public void WriteSbyte(sbyte val)
```

Parameters

- **val** The signed byte value to write to the message body.

Remarks

The signed byte is represented as a one byte value.

See also

- "QABinaryMessage interface [QAnywhere .NET]" on page 191
- "QABinaryMessage.ReadSbyte method [QAnywhere .NET]" on page 200

WriteShort method

Appends a short value to the QABinaryMessage instance's message body.

Visual Basic syntax

```
Public Sub WriteShort(ByVal val As Short)
```

C# syntax

```
public void WriteShort(short val)
```

Parameters

- **val** The short value to write to the message body.

Remarks

The short parameter is represented as a two byte value and the higher order byte is appended first.

See also

- "QABinaryMessage interface [QAnywhere .NET]" on page 191
- "QABinaryMessage.ReadShort method [QAnywhere .NET]" on page 200

WriteString method

Appends a string value to the QABinaryMessage instance's message body.

Visual Basic syntax

```
Public Sub WriteString(ByVal val As String)
```

C# syntax

```
public void WriteString(string val)
```

Parameters

- **val** The string value to write to the message body.

Remarks

Note
The receiving application needs to invoke QABinaryMessage.ReadString for each WriteString invocation.

Note
The UTF-8 representation of the string to be written can be at most 32767 bytes.

See also

- "QABinaryMessage interface [QAnywhere .NET]" on page 191
- "QABinaryMessage.ReadString method [QAnywhere .NET]" on page 201

BodyLength property

Returns the size of the message body in bytes.

Visual Basic syntax

```
Public ReadOnly Property BodyLength As Long
```

C# syntax

```
public long BodyLength {get;}
```

See also

- "QABinaryMessage interface [QAnywhere .NET]" on page 191

QAException class

Encapsulates QAnywhere client application exceptions.

Visual Basic syntax

```
Public MustInherit Class QAException Inherits System.Exception
```

C# syntax

```
public abstract class QAException : System.Exception
```

Base classes

- System.Exception

Members

All members of QAException class, including all inherited members.

Name	Description
GetBaseException method (Inherited from System.Exception)	When overridden in a derived class, returns the System.Exception that is the root cause of one or more subsequent exceptions.
GetObjectData method (Inherited from System.Exception)	When overridden in a derived class, sets the System.Runtime.Serialization.SerializationInfo with information about the exception.
GetType method (Inherited from System.Exception)	Gets the runtime type of the current instance.
ToString method (Inherited from System.Exception)	Creates and returns a string representation of the current exception.
Data property (Inherited from System.Exception)	Gets a collection of key/value pairs that provide additional user-defined information about the exception.
DetailedMessage property	The detailed error message of the exception.
ErrorCode property	The error code of the exception.
HelpLink property (Inherited from System.Exception)	Gets or sets a link to the help file associated with this exception.
HResult property (Inherited from System.Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.
InnerException property (Inherited from System.Exception)	Gets the System.Exception instance that caused the current exception.

Name	Description
Message property (Inherited from System.Exception)	Gets a message that describes the current exception.
NativeErrorCode property	The native error code of the exception.
Source property (Inherited from System.Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace property (Inherited from System.Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite property (Inherited from System.Exception)	Gets the method that throws the current exception.
COMMON_ALREADY_OPEN_ERROR field	The QAManager is already open.
COMMON_GET_INIT_FILE_ERROR field	Unable to access the client properties file.
COMMON_GET_PROPERTY_ERROR field	Error retrieving property from message store.
COMMON_GETQUEUEDEPTH_ERROR field	Error getting the queue depth.
COMMON_GETQUEUEDEPTH_ERROR_INVALID_ARG field	Cannot use QAManagerBase.getQueueDepth on a given destination when filter is ALL.
COMMON_GETQUEUEDEPTH_ERROR_NO_STORE_ID field	Cannot use QAManagerBase.getQueueDepth when the message store ID has not been set.
COMMON_INIT_ERROR field	Initialization error.
COMMON_INIT_THREAD_ERROR field	Error initializing the background thread.
COMMON_INVALID_PROPERTY field	There is an invalid property in the client properties file.
COMMON_MSG_ACKNOWLEDGE_ERROR field	Error acknowledging the message.
COMMON_MSG_CANCEL_ERROR field	Error canceling message.
COMMON_MSG_CANCEL_ERROR_SENT field	Error canceling message.
COMMON_MSG_NOT_WRITEABLE_ERROR field	You cannot write to a message that is in read-only mode.
COMMON_MSG_RETRIEVE_ERROR field	Error retrieving a message from the client message store.

Name	Description
COMMON_MSG_STORE_ERROR field	Error storing a message in the client message store.
COMMON_MSG_STORE_NOT_INITIALIZED field	The message store has not been initialized for messaging.
COMMON_MSG_STORE_TOO_LARGE field	The message store is too large relative to the free disk space on the device.
COMMON_NO_DEST_ERROR field	No destination.
COMMON_NO_IMPLEMENTATION field	The method is not implemented.
COMMON_NOT_OPEN_ERROR field	The QAManager is not open.
COMMON_OPEN_ERROR field	Error opening a connection to the message store.
COMMON_OPEN_LOG_FILE_ERROR field	Error opening the log file.
COMMON_OPEN_MAXTHREADS_ERROR field	Cannot open the QAManager because the maximum number of concurrent server requests is not high enough (see database server -gn option).
COMMON_REOPEN_ERROR field	Error re-opening connection to message store.
COMMON_SELECTOR_SYNTAX_ERROR field	The given selector has a syntax error.
COMMON_SET_PROPERTY_ERROR field	Error storing property to message store.
COMMON_TERMINATE_ERROR field	Termination error.
COMMON_UNEXPECTED_EOM_ERROR field	Unexpected end of message reached.
COMMON_UNREPRESENTABLE_TIMESTAMP field	The timestamp is outside of the acceptable range.
QA_NO_ERROR field	No error.

Remarks

You can use the QAException class to catch QAnywhere exceptions.

Example

The following method uses the QAException class to catch QAnywhere exceptions:

```
public static void startReceiver() {
    _mainWindow._messageList.Items.Clear();
    _mainWindow._detailWindow.Text = "";

    try {
          _qaManager = QAManagerFactory.Instance.CreateQAManager(null);
          _qaManager.Open(AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT);
          _qaManager.Start();
          _mainWindow.loadMessages();

_qaManager.SetMessageListener(Options.getReceiveQueueName(),_receiveListener)
;
          _qaManager.SetMessageListener("system", _systemListener);
    }
    catch(QAException e) {
          MessageBox.Show("Error code: " + e.ErrorCode );
          MessageBox.Show("Error message: " + e.Message );
    }
}
```

DetailedMessage property

The detailed error message of the exception.

Visual Basic syntax

```
Public ReadOnly Property DetailedMessage As String
```

C# syntax

```
public abstract string DetailedMessage {get;}
```

ErrorCode property

The error code of the exception.

Visual Basic syntax

```
Public ReadOnly Property ErrorCode As Integer
```

C# syntax

```
public abstract int ErrorCode {get;}
```

NativeErrorCode property

The native error code of the exception.

Visual Basic syntax

```
Public ReadOnly Property NativeErrorCode As Integer
```


C# syntax

```
public abstract int NativeErrorCode {get;}
```

COMMON_ALREADY_OPEN_ERROR field

The QAManager is already open.

Visual Basic syntax

```
Public Const COMMON_ALREADY_OPEN_ERROR As Integer
```

C# syntax

```
public const int COMMON_ALREADY_OPEN_ERROR;
```

See also

- "QAManager interface [QAnywhere .NET]" on page 221

COMMON_GET_INIT_FILE_ERROR field

Unable to access the client properties file.

Visual Basic syntax

```
Public Const COMMON_GET_INIT_FILE_ERROR As Integer
```

C# syntax

```
public const int COMMON_GET_INIT_FILE_ERROR;
```

COMMON_GET_PROPERTY_ERROR field

Error retrieving property from message store.

Visual Basic syntax

```
Public Const COMMON_GET_PROPERTY_ERROR As Integer
```

C# syntax

```
public const int COMMON_GET_PROPERTY_ERROR;
```

COMMON_GETQUEUEDEPTH_ERROR field

Error getting the queue depth.

Visual Basic syntax

```
Public Const COMMON_GETQUEUEDEPTH_ERROR As Integer
```

C# syntax

```
public const int COMMON_GETQUEUEDEPTH_ERROR;
```

See also

- "QAManagerBase.GetQueueDepth method [QAnywhere .NET]" on page 246

COMMON_GETQUEUEDEPTH_ERROR_INVALID_ARG field

Cannot use QAManagerBase.getQueueDepth on a given destination when filter is ALL.

Visual Basic syntax

```
Public Const COMMON_GETQUEUEDEPTH_ERROR_INVALID_ARG As Integer
```

C# syntax

```
public const int COMMON_GETQUEUEDEPTH_ERROR_INVALID_ARG;
```

See also

- "QAManagerBase.GetQueueDepth method [QAnywhere .NET]" on page 246

COMMON_GETQUEUEDEPTH_ERROR_NO_STORE_ID field

Cannot use QAManagerBase.getQueueDepth when the message store ID has not been set.

Visual Basic syntax

```
Public Const COMMON_GETQUEUEDEPTH_ERROR_NO_STORE_ID As Integer
```

C# syntax

```
public const int COMMON_GETQUEUEDEPTH_ERROR_NO_STORE_ID;
```

See also

- "QAManagerBase.GetQueueDepth method [QAnywhere .NET]" on page 246

COMMON_INIT_ERROR field

Initialization error.

Visual Basic syntax

```
Public Const COMMON_INIT_ERROR As Integer
```


C# syntax

```
public const int COMMON_INIT_ERROR;
```

COMMON_INIT_THREAD_ERROR field

Error initializing the background thread.

Visual Basic syntax

```
Public Const COMMON_INIT_THREAD_ERROR As Integer
```

C# syntax

```
public const int COMMON_INIT_THREAD_ERROR;
```

COMMON_INVALID_PROPERTY field

There is an invalid property in the client properties file.

Visual Basic syntax

```
Public Const COMMON_INVALID_PROPERTY As Integer
```

C# syntax

```
public const int COMMON_INVALID_PROPERTY;
```

COMMON_MSG_ACKNOWLEDGE_ERROR field

Error acknowledging the message.

Visual Basic syntax

```
Public Const COMMON_MSG_ACKNOWLEDGE_ERROR As Integer
```

C# syntax

```
public const int COMMON_MSG_ACKNOWLEDGE_ERROR;
```

COMMON_MSG_CANCEL_ERROR field

Error canceling message.

Visual Basic syntax

```
Public Const COMMON_MSG_CANCEL_ERROR As Integer
```

C# syntax

```
public const int COMMON_MSG_CANCEL_ERROR;
```

COMMON_MSG_CANCEL_ERROR_SENT field

Error canceling message.

Visual Basic syntax

```
Public Const COMMON_MSG_CANCEL_ERROR_SENT As Integer
```

C# syntax

```
public const int COMMON_MSG_CANCEL_ERROR_SENT;
```

Remarks

You cannot cancel a message that has already been sent.

COMMON_MSG_NOT_WRITEABLE_ERROR field

You cannot write to a message that is in read-only mode.

Visual Basic syntax

```
Public Const COMMON_MSG_NOT_WRITEABLE_ERROR As Integer
```

C# syntax

```
public const int COMMON_MSG_NOT_WRITEABLE_ERROR;
```

COMMON_MSG_RETRIEVE_ERROR field

Error retrieving a message from the client message store.

Visual Basic syntax

```
Public Const COMMON_MSG_RETRIEVE_ERROR As Integer
```

C# syntax

```
public const int COMMON_MSG_RETRIEVE_ERROR;
```

COMMON_MSG_STORE_ERROR field

Error storing a message in the client message store.

Visual Basic syntax

```
Public Const COMMON_MSG_STORE_ERROR As Integer
```

C# syntax

```
public const int COMMON_MSG_STORE_ERROR;
```

COMMON_MSG_STORE_NOT_INITIALIZED field

The message store has not been initialized for messaging.

Visual Basic syntax

```
Public Const COMMON_MSG_STORE_NOT_INITIALIZED As Integer
```

C# syntax

```
public const int COMMON_MSG_STORE_NOT_INITIALIZED;
```

COMMON_MSG_STORE_TOO_LARGE field

The message store is too large relative to the free disk space on the device.

Visual Basic syntax

```
Public Const COMMON_MSG_STORE_TOO_LARGE As Integer
```

C# syntax

```
public const int COMMON_MSG_STORE_TOO_LARGE;
```

COMMON_NO_DEST_ERROR field

No destination.

Visual Basic syntax

```
Public Const COMMON_NO_DEST_ERROR As Integer
```

C# syntax

```
public const int COMMON_NO_DEST_ERROR;
```

COMMON_NO_IMPLEMENTATION field

The method is not implemented.

Visual Basic syntax

```
Public Const COMMON_NO_IMPLEMENTATION As Integer
```

C# syntax

```
public const int COMMON_NO_IMPLEMENTATION;
```

COMMON_NOT_OPEN_ERROR field

The QAManager is not open.

Visual Basic syntax

```
Public Const COMMON_NOT_OPEN_ERROR As Integer
```

C# syntax

```
public const int COMMON_NOT_OPEN_ERROR;
```

See also

- "QAManager interface [QAnywhere .NET]" on page 221

COMMON_OPEN_ERROR field

Error opening a connection to the message store.

Visual Basic syntax

```
Public Const COMMON_OPEN_ERROR As Integer
```

C# syntax

```
public const int COMMON_OPEN_ERROR;
```

COMMON_OPEN_LOG_FILE_ERROR field

Error opening the log file.

Visual Basic syntax

```
Public Const COMMON_OPEN_LOG_FILE_ERROR As Integer
```

C# syntax

```
public const int COMMON_OPEN_LOG_FILE_ERROR;
```


COMMON_OPEN_MAXTHREADS_ERROR field

Cannot open the QAManager because the maximum number of concurrent server requests is not high enough (see database server -gn option).

Visual Basic syntax

```
Public Const COMMON_OPEN_MAXTHREADS_ERROR As Integer
```

C# syntax

```
public const int COMMON_OPEN_MAXTHREADS_ERROR;
```

COMMON_REOPEN_ERROR field

Error re-opening connection to message store.

Visual Basic syntax

```
Public Const COMMON_REOPEN_ERROR As Integer
```

C# syntax

```
public const int COMMON_REOPEN_ERROR;
```

COMMON_SELECTOR_SYNTAX_ERROR field

The given selector has a syntax error.

Visual Basic syntax

```
Public Const COMMON_SELECTOR_SYNTAX_ERROR As Integer
```

C# syntax

```
public const int COMMON_SELECTOR_SYNTAX_ERROR;
```

COMMON_SET_PROPERTY_ERROR field

Error storing property to message store.

Visual Basic syntax

```
Public Const COMMON_SET_PROPERTY_ERROR As Integer
```

C# syntax

```
public const int COMMON_SET_PROPERTY_ERROR;
```

COMMON_TERMINATE_ERROR field

Termination error.

Visual Basic syntax

```
Public Const COMMON_TERMINATE_ERROR As Integer
```

C# syntax

```
public const int COMMON_TERMINATE_ERROR;
```

COMMON_UNEXPECTED_EOM_ERROR field

Unexpected end of message reached.

Visual Basic syntax

```
Public Const COMMON_UNEXPECTED_EOM_ERROR As Integer
```

C# syntax

```
public const int COMMON_UNEXPECTED_EOM_ERROR;
```

COMMON_UNREPRESENTABLE_TIMESTAMP field

The timestamp is outside of the acceptable range.

Visual Basic syntax

```
Public Const COMMON_UNREPRESENTABLE_TIMESTAMP As Integer
```

C# syntax

```
public const int COMMON_UNREPRESENTABLE_TIMESTAMP;
```

QA_NO_ERROR field

No error.

Visual Basic syntax

```
Public Const QA_NO_ERROR As Integer
```

C# syntax

```
public const int QA_NO_ERROR;
```


QAManager interface

The QAManager class derives from QAManagerBase and manages non-transactional QAnywhere messaging operations.

Visual Basic syntax

```
Public Interface QAManager Inherits QAManagerBase
```

C# syntax

```
public interface QAManager : QAManagerBase
```

Base classes

- "QAManagerBase interface [QAnywhere .NET]" on page 228

Members

All members of QAManager interface, including all inherited members.

Name	Description
Acknowledge method	Acknowledges that the client application successfully received a QAnywhere message.
AcknowledgeAll method	Acknowledges that the client application successfully received QAnywhere messages.
AcknowledgeUntil method	Acknowledges the given QAMessage instance and all unacknowledged messages received before the given message.
BrowseMessages method	This method is deprecated.
BrowseMessagesByID method	Browses the message with the given message ID.
BrowseMessagesByQueue method	Browses the next available messages waiting that have been sent to the given address.
BrowseMessagesBySelector method	Browses messages queued in the message store that satisfy the given selector.
CancelMessage method	Cancels the message with the given message ID.
Close method	Closes the connection to the QAnywhere message system and releases any resources used by the QAManagerBase.
CreateBinaryMessage method	Creates a QABinaryMessage object.

Name	Description
CreateTextMessage method	Creates a QATextMessage object.
GetBooleanStoreProperty method	Gets a boolean value for a predefined or custom message store property.
GetDoubleStoreProperty method	Gets a double value for a predefined or custom message store property.
GetFloatStoreProperty method	Gets a float value for a predefined or custom message store property.
GetIntStoreProperty method	Gets an int value for a predefined or custom message store property.
GetLongStoreProperty method	Gets a long value for a predefined or custom message store property.
GetMessage method	Returns the next available QAMessage sent to the specified address.
GetMessageBySelector method	Returns the next available QAMessage sent to the specified address that satisfies the given selector.
GetMessageBySelectorNoWait method	Returns the next available QAMessage sent to the given address that satisfies the given selector.
GetMessageBySelectorTimeout method	Returns the next available QAMessage sent to the given address that satisfies the given selector.
GetMessageNoWait method	Returns the next available QAMessage sent to the given address.
GetMessageTimeout method	Returns the next available QAMessage sent to the given address.
GetQueueDepth method	Returns the total depth of all queues, based on a given filter.
GetSbyteStoreProperty method	Gets a signed byte value for a predefined or custom message store property.
GetShortStoreProperty method	Gets a short value for a predefined or custom message store property.
GetStoreProperty method	Gets a System.Object representing a message store property.
GetStorePropertyNames method	Gets an enumerator over the message store property names.
GetStringStoreProperty method	Gets a string value for a predefined or custom message store property.

Name	Description
Open method	Open the QAManager with the given AcknowledgementMode value.
PropertyExists method	Tests if there currently exists a value for the given the property.
PutMessage method	Prepares a message to send to another QAnywhere client.
PutMessageTimeToLive method	Prepares a message to send to another QAnywhere client.
Recover method	Forces all unacknowledged messages into a status of StatusCodes.PENDING.
ReOpen method	Reopens the QAManagerBase.
SetBooleanStoreProperty method	Sets a predefined or custom message store property to a boolean value.
SetDoubleStoreProperty method	Sets a predefined or custom message store property to a double value.
SetExceptionListener method	Sets an ExceptionListener delegate to receive QAExceptions when processing QAnywhere messages asynchronously.
SetExceptionListener2 method	Sets an ExceptionListener2 delegate to receive QAExceptions when processing QAnywhere messages asynchronously.
SetFloatStoreProperty method	Sets a predefined or custom message store property to a float value.
SetIntStoreProperty method	Sets a predefined or custom message store property to an int value.
SetLongStoreProperty method	Sets a predefined or custom message store property to a long value.
SetMessageListener method	Sets a MessageListener delegate to receive QAnywhere messages asynchronously.
SetMessageListener2 method	Sets a MessageListener2 delegate to receive QAnywhere messages asynchronously.
SetMessageListenerBySelector method	Sets a MessageListener delegate to receive QAnywhere messages asynchronously, with a message selector.
SetMessageListenerBySelector2 method	Sets a MessageListener2 delegate to receive QAnywhere messages asynchronously, with a message selector.

Name	Description
SetProperty method	Allows you to set QAnywhere Manager configuration properties programmatically.
SetSbyteStoreProperty method	Sets a predefined or custom message store property to an sbyte (signed byte) value.
SetShortStoreProperty method	Sets a predefined or custom message store property to a short value.
SetStoreProperty method	Sets a predefined or custom message store property to a System.Object value.
SetStringStoreProperty method	Sets a predefined or custom message store property to a string value.
Start method	Starts the QAManagerBase for receiving incoming messages in message listeners.
Stop method	Stops the QAManagerBase's reception of incoming messages.
TriggerSendReceive method	Causes a synchronization with the QAnywhere message server, uploading any messages addressed to other clients, and downloading any messages addressed to the local client.
Mode property	Returns the QAManager acknowledgement mode for received messages.

Remarks

For a detailed description of derived behavior, see QAManagerBase.

The QAManager can be configured for implicit or explicit acknowledgement as defined in the AcknowledgementMode class. To acknowledge messages as part of a transaction, use QATransactionalManager. Use the QAManagerFactory to create QAManager and QATransactionalManager objects.

See also

- "AcknowledgementMode enumeration [QAnywhere .NET]" on page 302
- "QATransactionalManager interface [QAnywhere .NET]" on page 295

Acknowledge method

Acknowledges that the client application successfully received a QAnywhere message.

Visual Basic syntax

```
Public Sub Acknowledge(ByVal msg As QAMessage)
```

C# syntax

```
public void Acknowledge(QAMessage msg)
```

Parameters

- **msg** the message to acknowledge.

Exceptions

- **QAException class** Thrown if there is a problem acknowledging the message.

Remarks

Note
when a QAMessage is acknowledged, its status property changes to StatusCodes.RECEIVED. When a QAMessage MessageProperties.STATUS message property changes to StatusCodes.RECEIVED, it can be deleted using the default delete rule.

For more information about delete rules, see "Message delete rules" on page 741.

See also

- "QAManager interface [QAnywhere .NET]" on page 221
- "QAManager.AcknowledgeUntil method [QAnywhere .NET]" on page 226
- "StatusCodes enumeration [QAnywhere .NET]" on page 306
- "MessageProperties class [QAnywhere .NET]" on page 181
- "QAManager.AcknowledgeAll method [QAnywhere .NET]" on page 225

AcknowledgeAll method

Acknowledges that the client application successfully received QAnywhere messages.

Visual Basic syntax

```
Public Sub AcknowledgeAll()
```

C# syntax

```
public void AcknowledgeAll()
```

Exceptions

- **QAException class** Thrown if there is a problem acknowledging the messages.

Remarks

All unacknowledged messages are acknowledged.

Note
when a QAMessage is acknowledged, its MessageProperties.STATUS property changes to StatusCodes.RECEIVED. When a QAMessage status changes to StatusCodes.RECEIVED, it can be deleted using the default delete rule.

For more information about delete rules, see "Message delete rules" on page 741.

See also

- "QAManager interface [QAnywhere .NET]" on page 221
- "QAManager.Acknowledge method [QAnywhere .NET]" on page 224
- "QAManager.AcknowledgeUntil method [QAnywhere .NET]" on page 226
- "StatusCodes enumeration [QAnywhere .NET]" on page 306
- "MessageProperties class [QAnywhere .NET]" on page 181

AcknowledgeUntil method

Acknowledges the given QAMessage instance and all unacknowledged messages received before the given message.

Visual Basic syntax

```
Public Sub AcknowledgeUntil(ByVal msg As QAMessage)
```

C# syntax

```
public void AcknowledgeUntil(QAMessage msg)
```

Parameters

- **msg** The last message to acknowledge. All earlier unacknowledged messages are also acknowledged.

Exceptions

- **QAException class** Thrown if there is a problem acknowledging the messages.

Remarks

Note
when a QAMessage is acknowledged, its MessageProperties.STATUS property changes to StatusCodes.RECEIVED. When a QAMessage status changes to StatusCodes.RECEIVED, it can be deleted using the default delete rule.

For more information about delete rules, see "Message delete rules" on page 741.

See also

- "QAManager interface [QAnywhere .NET]" on page 221
- "QAManager.Acknowledge method [QAnywhere .NET]" on page 224
- "QAManager.AcknowledgeAll method [QAnywhere .NET]" on page 225
- "StatusCodes enumeration [QAnywhere .NET]" on page 306
- "MessageProperties class [QAnywhere .NET]" on page 181

Open method

Open the QAManager with the given AcknowledgementMode value.

Visual Basic syntax

```
Public Sub Open(ByVal mode As AcknowledgementMode)
```

C# syntax

```
public void Open(AcknowledgementMode mode)
```

Parameters

- **mode** The acknowledgement mode, one of AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT or AcknowledgementMode.IMPLICIT_ACKNOWLEDGEMENT.

Exceptions

- **QAException class** Thrown if there is a problem opening the QAManager instance.

Remarks

The Open method must be the first method called after creating a QAManager.

See also

- "QAManager interface [QAnywhere .NET]" on page 221

Recover method

Forces all unacknowledged messages into a status of StatusCodes.PENDING.

Visual Basic syntax

```
Public Sub Recover()
```

C# syntax

```
public void Recover()
```

Exceptions

- **QAException class** Thrown if there is a problem recovering.

Remarks

That is, these messages must be received again using QAManagerBase.GetMessage.

See also

- "QAManager interface [QAnywhere .NET]" on page 221
- "QAManagerBase.GetMessage method [QAnywhere .NET]" on page 241

QAManagerBase interface

This class acts as a base class for QATransactionalManager and QAManager, which manage transactional and non-transactional messaging, respectively.

Visual Basic syntax

```
Public Interface QAManagerBase
```

C# syntax

```
public interface QAManagerBase
```

Derived classes

- "QAManager interface [QAnywhere .NET]" on page 221
- "QATransactionalManager interface [QAnywhere .NET]" on page 295

Members

All members of QAManagerBase interface, including all inherited members.

Name	Description
BrowseMessages method	This method is deprecated.
BrowseMessagesByID method	Browses the message with the given message ID.
BrowseMessagesByQueue method	Browses the next available messages waiting that have been sent to the given address.
BrowseMessagesBySelector method	Browses messages queued in the message store that satisfy the given selector.
CancelMessage method	Cancels the message with the given message ID.
Close method	Closes the connection to the QAnywhere message system and releases any resources used by the QAManagerBase.
CreateBinaryMessage method	Creates a QABinaryMessage object.

Name	Description
CreateTextMessage method	Creates a QATextMessage object.
GetBooleanStoreProperty method	Gets a boolean value for a predefined or custom message store property.
GetDoubleStoreProperty method	Gets a double value for a predefined or custom message store property.
GetFloatStoreProperty method	Gets a float value for a predefined or custom message store property.
GetIntStoreProperty method	Gets an int value for a predefined or custom message store property.
GetLongStoreProperty method	Gets a long value for a predefined or custom message store property.
GetMessage method	Returns the next available QAMessage sent to the specified address.
GetMessageBySelector method	Returns the next available QAMessage sent to the specified address that satisfies the given selector.
GetMessageBySelectorNoWait method	Returns the next available QAMessage sent to the given address that satisfies the given selector.
GetMessageBySelectorTimeout method	Returns the next available QAMessage sent to the given address that satisfies the given selector.
GetMessageNoWait method	Returns the next available QAMessage sent to the given address.
GetMessageTimeout method	Returns the next available QAMessage sent to the given address.
GetQueueDepth method	Returns the total depth of all queues, based on a given filter.
GetSbyteStoreProperty method	Gets a signed byte value for a predefined or custom message store property.
GetShortStoreProperty method	Gets a short value for a predefined or custom message store property.
GetStoreProperty method	Gets a System.Object representing a message store property.
GetStorePropertyNames method	Gets an enumerator over the message store property names.
GetStringStoreProperty method	Gets a string value for a predefined or custom message store property.

Name	Description
PropertyExists method	Tests if there currently exists a value for the given the property.
PutMessage method	Prepares a message to send to another QAnywhere client.
PutMessageTimeToLive method	Prepares a message to send to another QAnywhere client.
ReOpen method	Reopens the QAManagerBase.
SetBooleanStoreProperty method	Sets a predefined or custom message store property to a boolean value.
SetDoubleStoreProperty method	Sets a predefined or custom message store property to a double value.
SetExceptionListener method	Sets an ExceptionListener delegate to receive QAExceptions when processing QAnywhere messages asynchronously.
SetExceptionListener2 method	Sets an ExceptionListener2 delegate to receive QAExceptions when processing QAnywhere messages asynchronously.
SetFloatStoreProperty method	Sets a predefined or custom message store property to a float value.
SetIntStoreProperty method	Sets a predefined or custom message store property to an int value.
SetLongStoreProperty method	Sets a predefined or custom message store property to a long value.
SetMessageListener method	Sets a MessageListener delegate to receive QAnywhere messages asynchronously.
SetMessageListener2 method	Sets a MessageListener2 delegate to receive QAnywhere messages asynchronously.
SetMessageListenerBySelector method	Sets a MessageListener delegate to receive QAnywhere messages asynchronously, with a message selector.
SetMessageListenerBySelector2 method	Sets a MessageListener2 delegate to receive QAnywhere messages asynchronously, with a message selector.
SetProperty method	Allows you to set QAnywhere Manager configuration properties programmatically.
SetSbyteStoreProperty method	Sets a predefined or custom message store property to an sbyte (signed byte) value.

Name	Description
SetShortStoreProperty method	Sets a predefined or custom message store property to a short value.
SetStoreProperty method	Sets a predefined or custom message store property to a System.Object value.
SetStringStoreProperty method	Sets a predefined or custom message store property to a string value.
Start method	Starts the QAManagerBase for receiving incoming messages in message listeners.
Stop method	Stops the QAManagerBase's reception of incoming messages.
TriggerSendReceive method	Causes a synchronization with the QAnywhere message server, uploading any messages addressed to other clients, and downloading any messages addressed to the local client.
Mode property	Returns the QAManager acknowledgement mode for received messages.

Remarks

Use the QAManagerBase.Start() method to allow a QAManagerBase instance to listen for messages. There must be only a single instance of QAManagerBase per thread in your application.

You can use instances of this class to create and manage QAnywhere messages. Use the QAManagerBase.CreateBinaryMessage() method and the QAManagerBase.CreateTextMessage() method to create appropriate QAMessage instances. QAMessage instances provide a variety of methods to set message content and properties.

To send QAnywhere messages, use the QAManagerBase.PutMessage method to place the addressed message in the local message store queue. The message is transmitted by the QAnywhere Agent based on its transmission policies or when you call QAManagerBase.TriggerSendReceive().

For more information about qaagent transmission policies, see “Determining when message transmission should occur on the client” on page 46.

Messages are released from memory when you close a QAManagerBase instance using the QAManagerBase.Close method.

QAManagerBase also provides methods to set and get message store properties.

For more information, see “Client message store properties” on page 26 and the MessageStoreProperties class.

See also

- “QAManagerBase.CreateBinaryMessage method [QAnywhere .NET]” on page 236
- “QAManagerBase.TriggerSendReceive method [QAnywhere .NET]” on page 265
- “QAManagerBase.Close method [QAnywhere .NET]” on page 236
- “QAException class [QAnywhere .NET]” on page 209

BrowseMessages method

This method is deprecated.

Overload list

Name	Description
BrowseMessages() method	Browses all available messages in the message store.
BrowseMessages(string) method	This method is deprecated.

BrowseMessages() method

Browses all available messages in the message store.

Visual Basic syntax

```
Public Function BrowseMessages() As System.Collections.IEnumerator
```

C# syntax

```
public System.Collections.IEnumerator BrowseMessages()
```

Returns

An enumerator over the available messages.

Remarks

The messages are just being browsed, so they cannot be acknowledged. Because browsing messages allocates native resources, you should call the Reset() method of the enumerator when you are done with it. If it is not called, the native resources will not be freed until this QAManagerBase object is freed.

Use QAManagerBase.GetMessage to receive messages so they can be acknowledged.

See also

- “QAManagerBase.BrowseMessagesByQueue method [QAnywhere .NET]” on page 234
- “QAManagerBase.BrowseMessagesByID method [QAnywhere .NET]” on page 233
- “QAManagerBase.BrowseMessages method [QAnywhere .NET]” on page 232

BrowseMessages(string) method

This method is deprecated.

Visual Basic syntax

```
Public Function BrowseMessages(
     ByVal address As String
) As System.Collections.IEnumerator
```

C# syntax

```
public System.Collections.IEnumerator BrowseMessages(string address)
```

Parameters

- **address** The address of the messages.

Returns

An enumerator over the available messages.

Remarks

Use the BrowseMessagesByQueue(string) method instead.

Browses the next available messages waiting that have been sent to a given address. The address parameter takes the form store-id\queue-name or queue-name. The messages are just being browsed, so they cannot be acknowledged.

Because browsing messages allocates native resources, you should call the Reset() method of the enumerator when you are done with it. If it is not called, the native resources will not be freed until this QAManagerBase object is freed.

Use QAManagerBase.GetMessage to receive messages so they can be acknowledged.

See also

- "QAManagerBase.BrowseMessagesByQueue method [QAnywhere .NET]" on page 234
- "QAManagerBase.BrowseMessagesByID method [QAnywhere .NET]" on page 233
- "QAManagerBase.BrowseMessagesBySelector method [QAnywhere .NET]" on page 235
- "QAManagerBase.BrowseMessages method [QAnywhere .NET]" on page 232

BrowseMessagesByID method

Browses the message with the given message ID.

Visual Basic syntax

```
Public Function BrowseMessagesByID(
     ByVal msgid As String
) As System.Collections.IEnumerator
```

C# syntax

```
public System.Collections.IEnumerator BrowseMessagesByID(string msgid)
```

Parameters

- **msgid** The message id of the message.

Returns

An enumerator containing 0 or 1 messages.

Remarks

The message is just being browsed, so it cannot be acknowledged. Because browsing messages allocates native resources, you should call the Reset() method of the enumerator when you are done with it. If it is not called, the native resources will not be freed until this QAManagerBase object is freed.

Use QAManagerBase.GetMessage to receive messages so they can be acknowledged.

See also

- "QAManagerBase.BrowseMessagesByQueue method [QAnywhere .NET]" on page 234
- "QAManagerBase.BrowseMessages method [QAnywhere .NET]" on page 232

BrowseMessagesByQueue method

Browses the next available messages waiting that have been sent to the given address.

Visual Basic syntax

```
Public Function BrowseMessagesByQueue(
     ByVal address As String
) As System.Collections.IEnumerator
```

C# syntax

```
public System.Collections.IEnumerator BrowseMessagesByQueue(
     string address
)
```

Parameters

- **address** The address of the messages.

Returns

An enumerator over the available messages.

Remarks

The messages are just being browsed, so they cannot be acknowledged. Because browsing messages allocates native resources, you should call the Reset() method of the enumerator when you are done with it. If it is not called, the native resources will not be freed until this QAManagerBase object is freed.

Use QAManagerBase.GetMessage to receive messages so they can be acknowledged.

See also

- "QAManagerBase.BrowseMessagesByID method [QAnywhere .NET]" on page 233
- "QAManagerBase.BrowseMessages method [QAnywhere .NET]" on page 232

BrowseMessagesBySelector method

Browses messages queued in the message store that satisfy the given selector.

Visual Basic syntax

```
Public Function BrowseMessagesBySelector(
     ByVal selector As String
) As System.Collections.IEnumerator
```

C# syntax

```
public System.Collections.IEnumerator BrowseMessagesBySelector(
     string selector
)
```

Parameters

- **selector** The selector.

Returns

An enumerator over the available messages.

Remarks

The message is just being browsed, so it cannot be acknowledged. Because browsing messages allocates native resources, you should call the Reset() method of the enumerator when you are done with it. If it is not called, the native resources will not be freed until this QAManagerBase object is freed.

Use QAManagerBase.GetMessage to receive messages so they can be acknowledged.

See also

- "QAManagerBase.BrowseMessagesByQueue method [QAnywhere .NET]" on page 234
- "QAManagerBase.BrowseMessages method [QAnywhere .NET]" on page 232
- "QAManagerBase.BrowseMessagesByID method [QAnywhere .NET]" on page 233

CancelMessage method

Cancels the message with the given message ID.

Visual Basic syntax

```
Public Sub CancelMessage(ByVal msgid As String)
```

C# syntax

```
public void CancelMessage(string msgid)
```

Parameters

- **msgid** The message ID of the message to cancel.

Exceptions

- **QAException class** Thrown if there is a problem canceling the message.

Remarks

CancelMessage puts a message into a canceled state before it is transmitted. With the default delete rules of the QAnywhere Agent, canceled messages will eventually be deleted from the message store.

CancelMessage will fail if the message is already in a final state, or if it has been transmitted to the central messaging server.

For more information about delete rules, see "Message delete rules" on page 741.

Close method

Closes the connection to the QAnywhere message system and releases any resources used by the QAManagerBase.

Visual Basic syntax

```
Public Sub Close()
```

C# syntax

```
public void Close()
```

Exceptions

- **QAException class** Thrown if there is a problem closing the QAManagerBase instance.

Remarks

This method cannot be called in a message/exception listener.

Additional calls to Close() following the first are ignored.

CreateBinaryMessage method

Creates a QABinaryMessage object.

Visual Basic syntax

```
Public Function CreateBinaryMessage() As QABinaryMessage
```


C# syntax

```
public QABinaryMessage CreateBinaryMessage()
```

Returns

A new QABinaryMessage instance.

Exceptions

- **QAException class** Thrown if there is a problem creating the message.

Remarks

A QABinaryMessage object is used to send a message containing a message body of uninterpreted bytes.

See also

- "QABinaryMessage interface [QAnywhere .NET]" on page 191

CreateTextMessage method

Creates a QATextMessage object.

Visual Basic syntax

```
Public Function CreateTextMessage() As QATextMessage
```

C# syntax

```
public QATextMessage CreateTextMessage()
```

Returns

A new QATextMessage instance.

Exceptions

- **QAException class** Thrown if there is a problem creating the message.

Remarks

A QATextMessage object is used to send a message containing a string message body.

See also

- "QATextMessage interface [QAnywhere .NET]" on page 291

GetBooleanStoreProperty method

Gets a boolean value for a predefined or custom message store property.

Visual Basic syntax

```
Public Function GetBooleanStoreProperty(
     ByVal propName As String
) As Boolean
```

C# syntax

```
public bool GetBooleanStoreProperty(string propName)
```

Parameters

- **propName** The predefined or custom property name.

Returns

The boolean property value.

Exceptions

- **QAException class** Thrown if there is a conversion error or message store error getting the property value or if the property does not exist.

Remarks

You can use this method to access predefined or user-defined client store properties.

For a list of predefined properties, see MessageStoreProperties.

For more information, see "Client message store properties" on page 26.

See also

- "MessageStoreProperties class [QAnywhere .NET]" on page 190

GetDoubleStoreProperty method

Gets a double value for a predefined or custom message store property.

Visual Basic syntax

```
Public Function GetDoubleStoreProperty(
     ByVal propName As String
) As Double
```

C# syntax

```
public double GetDoubleStoreProperty(string propName)
```

Parameters

- **propName** The predefined or custom property name.

Returns

The double property value.

Exceptions

- **QAException class** Thrown if there is a conversion error or message store error getting the property value or if the property does not exist.

Remarks

You can use this method to access predefined or user-defined client store properties.

For a list of predefined properties, see MessageStoreProperties.

For more information, see "Client message store properties" on page 26.

See also

- "MessageStoreProperties class [QAnywhere .NET]" on page 190

GetFloatStoreProperty method

Gets a float value for a predefined or custom message store property.

Visual Basic syntax

```
Public Function GetFloatStoreProperty(
     ByVal propName As String
) As Single
```

C# syntax

```
public float GetFloatStoreProperty(string propName)
```

Parameters

- **propName** The predefined or custom property name.

Returns

The float property value.

Exceptions

- **QAException class** Thrown if there is a conversion error or message store error getting the property value or if the property does not exist.

Remarks

You can use this method to access predefined or user-defined client store properties.

For a list of predefined properties, see MessageStoreProperties.

For more information, see "Client message store properties" on page 26.

See also

- "MessageStoreProperties class [QAnywhere .NET]" on page 190

GetIntStoreProperty method

Gets an int value for a predefined or custom message store property.

Visual Basic syntax

```
Public Function GetIntStoreProperty(ByVal propName As String) As Integer
```

C# syntax

```
public int GetIntStoreProperty(string propName)
```

Parameters

- **propName** The predefined or custom property name.

Returns

The integer property value.

Exceptions

- **QAException class** Thrown if there is a conversion error or message store error getting the property value or if the property does not exist.

Remarks

You can use this method to access predefined or user-defined client store properties.

For a list of predefined properties, see MessageStoreProperties.

For more information, see "Client message store properties" on page 26.

See also

- "MessageStoreProperties class [QAnywhere .NET]" on page 190

GetLongStoreProperty method

Gets a long value for a predefined or custom message store property.

Visual Basic syntax

```
Public Function GetLongStoreProperty(ByVal propName As String) As Long
```

C# syntax

```
public long GetLongStoreProperty(string propName)
```

Parameters

- **propName** The predefined or custom property name.

Returns

The long property value.

Exceptions

- **QAException class** Thrown if there is a conversion error or message store error getting the property value or if the property does not exist.

Remarks

You can use this method to access predefined or user-defined client store properties.

For a list of predefined properties, see MessageStoreProperties.

For more information, see "Client message store properties" on page 26

See also

- "MessageStoreProperties class [QAnywhere .NET]" on page 190

GetMessage method

Returns the next available QAMessage sent to the specified address.

Visual Basic syntax

```
Public Function GetMessage(ByVal address As String) As QAMessage
```

C# syntax

```
public QAMessage GetMessage(string address)
```

Parameters

- **address** Specifies the queue name used by the QAnywhere client to receive messages.

Returns

The next QAMessage, or null if no message is available.

Exceptions

- **QAException class** Thrown if there is a problem getting the message.

Remarks

The address parameter specifies a local queue name. The address can be in the form store-id\queue-name or queue-name.

If there is no message available, this call blocks indefinitely until a message is available. Use this method to receive messages synchronously.

For more information about receiving messages asynchronously (using a message event handler), see "Receiving messages asynchronously" on page 69.

See also

- "QAMessage interface [QAnywhere .NET]" on page 272

GetMessageBySelector method

Returns the next available QAMessage sent to the specified address that satisfies the given selector.

Visual Basic syntax

```
Public Function GetMessageBySelector(
     ByVal address As String,
    ByVal selector As String
) As QAMessage
```

C# syntax

```
public QAMessage GetMessageBySelector(string address, string selector)
```

Parameters

- **address** This address specifies the queue name used by the QAnywhere client to receive messages.
- **selector** The selector.

Returns

The next QAMessage, or null if no message is available.

Exceptions

- **QAException class** Thrown if there is a problem getting the message.

Remarks

The address parameter specifies a local queue name. The address can be in the form store-id\queue-name or queue-name.

If there is no message available, this call blocks indefinitely until a message is available. Use this method to receive messages synchronously.

For more information about receiving messages asynchronously (using a message event handler), see "Receiving messages asynchronously" on page 69.

See also

- "QAMessage interface [QAnywhere .NET]" on page 272

GetMessageBySelectorNoWait method

Returns the next available QAMessage sent to the given address that satisfies the given selector.

Visual Basic syntax

```
Public Function GetMessageBySelectorNoWait(
     ByVal address As String,
    ByVal selector As String
) As QAMessage
```

C# syntax

```
public QAMessage GetMessageBySelectorNoWait(
     string address,
    string selector
)
```

Parameters

- **address** Specifies the queue name used by the QAnywhere client to receive messages.
- **selector** The selector.

Returns

The next available message or null there are no available message.

Exceptions

- **QAException class** Thrown if there is a problem getting the message.

Remarks

The address parameter specifies a local queue name. The address can be in the form store-id\queue-name or queue-name. If no message is available, this method returns immediately. Use this method to receive messages synchronously.

For more information about receiving messages asynchronously (using a message event handler), see "Receiving messages asynchronously" on page 69.

See also

- "QAMessage interface [QAnywhere .NET]" on page 272

GetMessageBySelectorTimeout method

Returns the next available QAMessage sent to the given address that satisfies the given selector.

Visual Basic syntax

```
Public Function GetMessageBySelectorTimeout(
     ByVal address As String,
    ByVal selector As String,
    ByVal timeout As Long
) As QAMessage
```

C# syntax

```
public QAMessage GetMessageBySelectorTimeout(
     string address,
    string selector,
    long timeout
)
```

Parameters

- **address** Specifies the queue name used by the QAnywhere client to receive messages.
- **selector** The selector.
- **timeout** The time to wait, in milliseconds, for a message to become available.

Returns

The next QAMessage, or null if no message is available.

Exceptions

- **QAException class** Thrown if there is a problem getting the message.

Remarks

The address parameter specifies a local queue name. The address can be in the form store-id\queue-name or queue-name. If no message is available, this method waits for the specified timeout and then returns. Use this method to receive messages synchronously.

For more information about receiving messages asynchronously (using a message event handler), see "Receiving messages asynchronously" on page 69.

See also

- "QAMessage interface [QAnywhere .NET]" on page 272

GetMessageNoWait method

Returns the next available QAMessage sent to the given address.

Visual Basic syntax

```
Public Function GetMessageNoWait(ByVal address As String) As QAMessage
```

C# syntax

```
public QAMessage GetMessageNoWait(string address)
```

Parameters

- **address** this address specifies the queue name used by the QAnywhere client to receive messages.

Returns

The next available message or null there is no available message.

Exceptions

- **QAException class** Thrown if there is a problem getting the message.

Remarks

The address parameter specifies a local queue name. The address can be in the form store-id\queue-name or queue-name. If no message is available, this method returns immediately. Use this method to receive messages synchronously. For more information about receiving messages asynchronously (using a message event handler), see "Receiving messages asynchronously" on page 69.

See also

- "QAMessage interface [QAnywhere .NET]" on page 272

GetMessageTimeout method

Returns the next available QAMessage sent to the given address.

Visual Basic syntax

```
Public Function GetMessageTimeout(
     ByVal address As String,
    ByVal timeout As Long
) As QAMessage
```

C# syntax

```
public QAMessage GetMessageTimeout(string address, long timeout)
```

Parameters

- **address** Specifies the queue name used by the QAnywhere client to receive messages.
- **timeout** The time to wait, in milliseconds, for a message to become available.

Returns

The next QAMessage, or null if no message is available.

Exceptions

- **QAException class** Thrown if there is a problem getting the message.

Remarks

The address parameter specifies a local queue name. The address can be in the form store-id\queue-name or queue-name.

If no message is available, this method waits for the specified timeout and then returns. Use this method to receive messages synchronously.

For more information about receiving messages asynchronously (using a message event handler), see "Receiving messages asynchronously" on page 69.

GetQueueDepth method

Returns the total depth of all queues, based on a given filter.

Overload list

Name	Description
GetQueueDepth(QueueDepthFilter) method	Returns the total depth of all queues, based on a given filter.
GetQueueDepth(string, QueueDepthFilter) method	Returns the depth of a queue, based on a given filter.

GetQueueDepth(QueueDepthFilter) method

Returns the total depth of all queues, based on a given filter.

Visual Basic syntax

```
Public Function GetQueueDepth(
     ByVal filter As QueueDepthFilter
) As Integer
```

C# syntax

```
public int GetQueueDepth(QueueDepthFilter filter)
```

Parameters

- **filter** A filter indicating incoming messages, outgoing messages, or all messages.

Returns

The number of messages.

Exceptions

- **QAException class** Thrown if there was an error.

Remarks

The incoming depth of the queue is the number of incoming messages which have not been received (for example, using QAManagerBase.GetMessage). The outgoing depth of a queue is the number of outgoing messages (including uncommitted) that have not been transmitted to the server.

See also

- "QueueDepthFilter enumeration [QAnywhere .NET]" on page 305

GetQueueDepth(string, QueueDepthFilter) method

Returns the depth of a queue, based on a given filter.

Visual Basic syntax

```
Public Function GetQueueDepth(
     ByVal address As String,
    ByVal filter As QueueDepthFilter
) As Integer
```

C# syntax

```
public int GetQueueDepth(string address, QueueDepthFilter filter)
```

Parameters

- **filter** A filter indicating incoming messages, outgoing messages, or all messages.
- **address** The queue name.

Returns

The number of messages.

Exceptions

- **QAException class** Thrown if there was an error.

Remarks

The incoming depth of the queue is the number of incoming messages which have not been received (for example, using QAManagerBase.GetMessage). The outgoing depth of a queue is the number of outgoing messages (including uncommitted) that have not been transmitted to the server.

See also

- "QueueDepthFilter enumeration [QAnywhere .NET]" on page 305

GetSbyteStoreProperty method

Gets a signed byte value for a predefined or custom message store property.

Visual Basic syntax

```
Public Function GetSbyteStoreProperty(ByVal propName As String) As SByte
```

C# syntax

```
public sbyte GetSbyteStoreProperty(string propName)
```

Parameters

- **propName** The predefined or custom property name.

Returns

The signed byte property value.

Exceptions

- **QAException class** Thrown if there is a conversion error or message store error getting the property value or if the property does not exist.

Remarks

You can use this method to access predefined or user-defined client store properties.

For a list of predefined properties, see MessageStoreProperties.

For more information, see "Client message store properties" on page 26.

See also

- "MessageStoreProperties class [QAnywhere .NET]" on page 190

GetShortStoreProperty method

Gets a short value for a predefined or custom message store property.

Visual Basic syntax

```
Public Function GetShortStoreProperty(ByVal propName As String) As Short
```

C# syntax

```
public short GetShortStoreProperty(string propName)
```

Parameters

- **propName** the predefined or custom property name.

Returns

The short property value.

Exceptions

- **QAException class** Thrown if there is a conversion error or message store error getting the property value or if the property does not exist.

Remarks

You can use this method to access predefined or user-defined client store properties.

For a list of predefined properties, see MessageStoreProperties.

For more information, see "Client message store properties" on page 26.

See also

- "MessageStoreProperties class [QAnywhere .NET]" on page 190

GetStoreProperty method

Gets a System.Object representing a message store property.

Visual Basic syntax

```
Public Function GetStoreProperty(ByVal propName As String) As Object
```

C# syntax

```
public Object GetStoreProperty(string propName)
```

Parameters

- **propName** The predefined or custom property name.

Returns

The property value.

Exceptions

- **QAException class** Thrown if there is a problem retrieving the property.

Remarks

You can use this method to access predefined or user-defined client store properties.

For a list of predefined properties, see MessageStoreProperties.

For more information, see "Client message store properties" on page 26.

See also

- "MessageStoreProperties class [QAnywhere .NET]" on page 190

GetStorePropertyNames method

Gets an enumerator over the message store property names.

Visual Basic syntax

```
Public Function GetStorePropertyNames()
    As System.Collections.IEnumerator
```

C# syntax

```
public System.Collections.IEnumerator GetStorePropertyNames()
```

Returns

An enumerator over the message store property names.

Remarks

For more information about client store properties, see "Client message store properties" on page 26.

GetStringStoreProperty method

Gets a string value for a predefined or custom message store property.

Visual Basic syntax

```
Public Function GetStringStoreProperty(
     ByVal propName As String
) As String
```

C# syntax

```
public string GetStringStoreProperty(string propName)
```

Parameters

- **propName** The predefined or custom property name.

Returns

The string property value or null if the property does not exist.

Exceptions

- **QAException class** Thrown if there is a problem retrieving the string value.

Remarks

You can use this method to access predefined or user-defined client store properties.

For a list of predefined properties, see MessageStoreProperties.

For more information, see "Client message store properties" on page 26.

See also

- "MessageStoreProperties class [QAnywhere .NET]" on page 190

PropertyExists method

Tests if there currently exists a value for the given the property.

Visual Basic syntax

```
Public Function PropertyExists(ByVal propName As String) As Boolean
```


C# syntax

```
public bool PropertyExists(string propName)
```

Parameters

- **propName** The predefined or custom property name.

Returns

true if the message store has a value mapped to the property. false otherwise.

Exceptions

- **QAException class** Thrown if there is a problem retrieving the property value.

Remarks

You can use this method to determine if a given property name currently has a value mapped to it by the message store.

See also

- "MessageStoreProperties class [QAnywhere .NET]" on page 190

PutMessage method

Prepares a message to send to another QAnywhere client.

Visual Basic syntax

```
Public Sub PutMessage(ByVal address As String, ByVal msg As QAMessage)
```

C# syntax

```
public void PutMessage(string address, QAMessage msg)
```

Parameters

- **address** The address of the message specifying the destination queue name.
- **msg** The message to put in the local message store for transmission.

Exceptions

- **QAException class** Thrown if there is a problem putting the message.

Remarks

The PutMessage method inserts a message and a destination address into your local message store. The time of message transmission depends on QAnywhere Agent transmission policies.

For more information, see "Determining when message transmission should occur on the client" on page 46.

The address takes the form id\queue-name, where id is the destination message store ID and queue-name identifies a queue that is used by the destination QAnywhere client to listen for or receive messages.

For more information about QAnywhere addresses, see "QAnywhere message addresses" on page 57.

See also

- "QAManagerBase.PutMessageTimeToLive method [QAnywhere .NET]" on page 252

PutMessageTimeToLive method

Prepares a message to send to another QAnywhere client.

Visual Basic syntax

```
Public Sub PutMessageTimeToLive(
     ByVal address As String,
    ByVal msg As QAMessage,
    ByVal ttl As Long
)
```

C# syntax

```
public void PutMessageTimeToLive(
     string address,
    QAMessage msg,
    long ttl
)
```

Parameters

- **address** The address of the message specifying the destination queue name.
- **msg** The message to put.
- **ttl** The delay, in milliseconds, before the message will expire if it has not been delivered. A value of 0 indicates the message will not expire.

Exceptions

- **QAException class** Thrown if there is a problem putting the message.

Remarks

The PutMessageTimeToLive method inserts a message and a destination address into your local message store. The time of message transmission depends on QAnywhere Agent transmission policies. However, if the next message transmission time exceeds the given time-to-live value, the message expires.

For more information, see "Determining when message transmission should occur on the client" on page 46.

The address takes the form id\queue-name, where id is the destination message store id and queue-name identifies a queue that is used by the destination QAnywhere client to listen for or receive messages.

For more information about QAnywhere addresses, see "QAnywhere message addresses" on page 57.

ReOpen method

Reopens the QAManagerBase.

Visual Basic syntax

```
Public Sub ReOpen()
```

C# syntax

```
public void ReOpen()
```

Exceptions

- **QAException class** Thrown if there is a problem reopening the QAManagerBase instance.

Remarks

This re-establishes connections to the message store, without releasing any resources. This method may called in a message or exception listener, and in that case it is not necessary to call Start() again. This method simply executes Close() then Open() if not called in a listener, and in that case Start() must be called to restart receiving of messages.

SetBooleanStoreProperty method

Sets a predefined or custom message store property to a boolean value.

Visual Basic syntax

```
Public Sub SetBooleanStoreProperty(
     ByVal propName As String,
    ByVal val As Boolean
)
```

C# syntax

```
public void SetBooleanStoreProperty(string propName, bool val)
```

Parameters

- **propName** The predefined or custom property name.
- **val** The boolean property value.

Remarks

You can use this method to set predefined or user-defined client store properties.

For a list of predefined properties, see MessageStoreProperties.

For more information, see "Client message store properties" on page 26.

See also

- "MessageStoreProperties class [QAnywhere .NET]" on page 190

SetDoubleStoreProperty method

Sets a predefined or custom message store property to a double value.

Visual Basic syntax

```
Public Sub SetDoubleStoreProperty(
     ByVal propName As String,
    ByVal val As Double
)
```

C# syntax

```
public void SetDoubleStoreProperty(string propName, double val)
```

Parameters

- **propName** The predefined or custom property name.
- **val** The double property value.

Remarks

You can use this method to set predefined or user-defined client. store properties.

For a list of predefined properties, see MessageStoreProperties.

For more information, see "Client message store properties" on page 26.

See also

- "MessageStoreProperties class [QAnywhere .NET]" on page 190

SetExceptionListener method

Sets an ExceptionListener delegate to receive QAExceptions when processing QAnywhere messages asynchronously.

Visual Basic syntax

```
Public Sub SetExceptionListener(
     ByVal address As String,
    ByVal listener As ExceptionListener
)
```

C# syntax

```
public void SetExceptionListener(
     string address,
```



```
    ExceptionListener listener
)
```

Parameters

- **address** The address of messages.
- **listener** The exception listener to register.

Remarks

ExceptionListener delegate accepts QAException and QAMessage parameters. You may set an ExceptionListener and a MessageListener for a given address, but you must be consistent with the Listener/Listener2 delegates. That is, you cannot set an ExceptionListener and a MessageListener2, nor an ExceptionListener2 and a MessageListener, for the same address.

For more information, see "Receiving messages asynchronously" on page 69.

SetExceptionListener2 method

Sets an ExceptionListener2 delegate to receive QAExceptions when processing QAnywhere messages asynchronously.

Visual Basic syntax

```
Public Sub SetExceptionListener2(
     ByVal address As String,
    ByVal listener As ExceptionListener2
)
```

C# syntax

```
public void SetExceptionListener2(
     string address,
    ExceptionListener2 listener
)
```

Parameters

- **address** The address of messages.
- **listener** The exception listener to register.

Remarks

ExceptionListener2 delegate accepts QAManagerBase, QAException and QAMessage parameters. You may set an ExceptionListener2 and a MessageListener2 for a given address, but you must be consistent with the Listener/Listener2 delegates. That is, you cannot set an ExceptionListener and a MessageListener2, nor an ExceptionListener2 and a MessageListener, for the same address.

For more information, see "Receiving messages asynchronously" on page 69.

SetFloatStoreProperty method

Sets a predefined or custom message store property to a float value.

Visual Basic syntax

```
Public Sub SetFloatStoreProperty(
     ByVal propName As String,
    ByVal val As Single
)
```

C# syntax

```
public void SetFloatStoreProperty(string propName, float val)
```

Parameters

- **propName** The predefined or custom property name.
- **val** The float property value.

Remarks

You can use this method to set predefined or user-defined client store properties.

For a list of predefined properties, see MessageStoreProperties.

For more information, see "Client message store properties" on page 26.

See also

- "MessageStoreProperties class [QAnywhere .NET]" on page 190

SetIntStoreProperty method

Sets a predefined or custom message store property to an int value.

Visual Basic syntax

```
Public Sub SetIntStoreProperty(
     ByVal propName As String,
    ByVal val As Integer
)
```

C# syntax

```
public void SetIntStoreProperty(string propName, int val)
```

Parameters

- **propName** The predefined or custom property name.
- **val** The int property value.

Remarks

You can use this method to set predefined or user-defined client store properties.

For a list of predefined properties, see MessageStoreProperties.

For more information, see "Client message store properties" on page 26.

See also

- "MessageStoreProperties class [QAnywhere .NET]" on page 190

SetLongStoreProperty method

Sets a predefined or custom message store property to a long value.

Visual Basic syntax

```
Public Sub SetLongStoreProperty(
     ByVal propName As String,
    ByVal val As Long
)
```

C# syntax

```
public void SetLongStoreProperty(string propName, long val)
```

Parameters

- **propName** The predefined or custom property name.
- **val** The long property value

Remarks

You can use this method to set predefined or user-defined client store properties.

For a list of predefined properties, see MessageStoreProperties.

For more information, see "Client message store properties" on page 26.

See also

- "MessageStoreProperties class [QAnywhere .NET]" on page 190

SetMessageListener method

Sets a MessageListener delegate to receive QAnywhere messages asynchronously.

Visual Basic syntax

```
Public Sub SetMessageListener(
     ByVal address As String,
```

```
    ByVal listener As MessageListener
)
```

C# syntax

```
public void SetMessageListener(string address, MessageListener listener)
```

Parameters

- **address** The address of messages.
- **listener** The listener to register.

Remarks

Use this method to receive message asynchronously.

MessageListener delegate accepts a single QAMessage parameter.

The SetMessageListener address parameter specifies a local queue name used to receive the message. You can only have one listener delegate assigned to a given queue. You may set an ExceptionListener and a MessageListener for a given address, but you must be consistent with the Listener/Listener2 delegates. That is, you cannot set an ExceptionListener and a MessageListener2, nor an ExceptionListener2 and a MessageListener, for the same address.

If you want to listen for QAnywhere system messages, including push notifications and network status changes, specify system as the queue name.

For more information, see "Receiving messages asynchronously" on page 69.

See also

- "MessageListener delegate [QAnywhere .NET]" on page 301

SetMessageListener2 method

Sets a MessageListener2 delegate to receive QAnywhere messages asynchronously.

Visual Basic syntax

```
Public Sub SetMessageListener2(
     ByVal address As String,
    ByVal listener As MessageListener2
)
```

C# syntax

```
public void SetMessageListener2(
     string address,
    MessageListener2 listener
)
```

Parameters

- **address** The address of messages.

- **listener** The listener to register.

Remarks

Use this method to receive message asynchronously.

MessageListener2 delegate accepts QAManagerBase and QAMessage parameters.

The SetMessageListener2 address parameter specifies a local queue name used to receive the message. You can only have one listener delegate assigned to a given queue. You may set an ExceptionListener2 and a MessageListener2 for a given address, but you must be consistent with the Listener/Listener2 delegates. That is, you cannot set an ExceptionListener and a MessageListener2, nor an ExceptionListener2 and a MessageListener, for the same address.

If you want to listen for QAnywhere system messages, including push notifications and network status changes, specify system as the queue name.

For more information, see "Receiving messages asynchronously" on page 69.

SetMessageListenerBySelector method

Sets a MessageListener delegate to receive QAnywhere messages asynchronously, with a message selector.

Visual Basic syntax

```
Public Sub SetMessageListenerBySelector(
     ByVal address As String,
    ByVal selector As String,
    ByVal listener As MessageListener
)
```

C# syntax

```
public void SetMessageListenerBySelector(
     string address,
    string selector,
    MessageListener listener
)
```

Parameters

- **address** The address of messages.
- **listener** The listener to register.
- **selector** The selector to be used to filter the messages to be received.

Remarks

Use this method to receive message asynchronously.

MessageListener delegate accepts a single QAMessage parameter.

The SetMessageListener address parameter specifies a local queue name used to receive the message. You can only have one listener delegate assigned to a given queue. The selector parameter specifies a selector to be used to filter the messages to be received on the given address. You may set an ExceptionListener and a MessageListener for a given address, but you must be consistent with the Listener/Listener2 delegates. That is, you cannot set an ExceptionListener and a MessageListener2, nor an ExceptionListener2 and a MessageListener, for the same address.

If you want to listen for QAnywhere system messages, including push notifications and network status changes, specify system as the queue name.

For more information, see "Receiving messages asynchronously" on page 69 and "System queue" on page 58.

See also

- "MessageListener delegate [QAnywhere .NET]" on page 301

SetMessageListenerBySelector2 method

Sets a MessageListener2 delegate to receive QAnywhere messages asynchronously, with a message selector.

Visual Basic syntax

```
Public Sub SetMessageListenerBySelector2(
     ByVal address As String,
    ByVal selector As String,
    ByVal listener As MessageListener2
)
```

C# syntax

```
public void SetMessageListenerBySelector2(
     string address,
    string selector,
    MessageListener2 listener
)
```

Parameters

- **address** The address of messages.
- **listener** The listener to register.
- **selector** The selector to be used to filter the messages to be received.

Remarks

Use this method to receive message asynchronously.

MessageListener2 delegate accepts a single QAMessage parameter.

The SetMessageListener2 address parameter specifies a local queue name used to receive the message. You can only have one listener delegate assigned to a given queue. The selector parameter specifies a

selector to be used to filter the messages to be received on the given address. You may set an ExceptionListener2 and a MessageListener2 for a given address, but you must be consistent with the Listener/Listener2 delegates. That is, you cannot set an ExceptionListener and a MessageListener2, nor an ExceptionListener2 and a MessageListener, for the same address.

If you want to listen for QAnywhere system messages, including push notifications and network status changes, specify system as the queue name.

For more information, see "Receiving messages asynchronously" on page 69 and "System queue" on page 58.

SetProperty method

Allows you to set QAnywhere Manager configuration properties programmatically.

Visual Basic syntax

```
Public Sub SetProperty(ByVal name As String, ByVal val As String)
```

C# syntax

```
public void SetProperty(string name, string val)
```

Parameters

- **name** The QAnywhere Manager configuration property name.
- **val** The QAnywhere Manager configuration property value

Exceptions

- **QAException class** Thrown if there is a problem setting the property.

Remarks

You can use this method to override default QAnywhere Manager configuration properties by specifying a property name and value. For a list of properties, see "QAnywhere manager configuration properties" on page 80.

You can also set QAnywhere Manager configuration properties using a properties file and the QAManagerFactory.CreateQAManager method.

> **Note**
> For more information, see "QAnywhere manager configuration property settings in a file" on page 82. you must set required properties before calling QAManager.Open or QATransactionalManager.Open().

See also

- "QAManager.Open method [QAnywhere .NET]" on page 227
- "QATransactionalManager.Open method [QAnywhere .NET]" on page 299

SetSbyteStoreProperty method

Sets a predefined or custom message store property to an sbyte (signed byte) value.

Visual Basic syntax

```
Public Sub SetSbyteStoreProperty(
     ByVal propName As String,
    ByVal val As SByte
)
```

C# syntax

```
public void SetSbyteStoreProperty(string propName, sbyte val)
```

Parameters

- **propName** The predefined or custom property name.
- **val** The sbyte (signed byte) property value.

Remarks

You can use this method to set predefined or user-defined client store properties.

For a list of predefined properties, see MessageStoreProperties.

For more information, see "Client message store properties" on page 26.

See also

- "MessageStoreProperties class [QAnywhere .NET]" on page 190

SetShortStoreProperty method

Sets a predefined or custom message store property to a short value.

Visual Basic syntax

```
Public Sub SetShortStoreProperty(
     ByVal propName As String,
    ByVal val As Short
)
```

C# syntax

```
public void SetShortStoreProperty(string propName, short val)
```

Parameters

- **propName** The predefined or custom property name.
- **val** The short property value.

Remarks

You can use this method to set predefined or user-defined client store properties.

For a list of predefined properties, see MessageStoreProperties.

For more information, see "Client message store properties" on page 26.

See also

- "MessageStoreProperties class [QAnywhere .NET]" on page 190

SetStoreProperty method

Sets a predefined or custom message store property to a System.Object value.

Visual Basic syntax

```
Public Sub SetStoreProperty(
     ByVal propName As String,
    ByVal val As Object
)
```

C# syntax

```
public void SetStoreProperty(string propName, Object val)
```

Parameters

- **propName** The predefined or custom property name.
- **val** The property value.

Remarks

The property type must be one of the acceptable simple types, string or DateTime. You can use this method to set predefined or user-defined client store properties.

For a list of predefined properties, see MessageStoreProperties.

For more information, see "Client message store properties" on page 26.

See also

- "MessageStoreProperties class [QAnywhere .NET]" on page 190

SetStringStoreProperty method

Sets a predefined or custom message store property to a string value.

Visual Basic syntax

```
Public Sub SetStringStoreProperty(
     ByVal propName As String,
```

```
        ByVal val As String
    )
```

C# syntax

```
public void SetStringStoreProperty(string propName, string val)
```

Parameters

- **propName** The predefined or custom property name.
- **val** The string property value.

Remarks

You can use this method to set predefined or user-defined client store properties.

For a list of predefined properties, see MessageStoreProperties.

For more information, see "Client message store properties" on page 26.

See also

- "MessageStoreProperties class [QAnywhere .NET]" on page 190

Start method

Starts the QAManagerBase for receiving incoming messages in message listeners.

Visual Basic syntax

```
Public Sub Start()
```

C# syntax

```
public void Start()
```

Exceptions

- **QAException class** Thrown if there is a problem starting the QAManagerBase instance.

Remarks

The QAManagerBase does not need to be started if there are no message listeners set, that is, if messages are received with the GetMessage methods. It is not recommended to use the GetMessage methods as well as message listeners for receiving messages. Use one or the other of the asynchronous (message listener) or synchronous (GetMessage) models. Any calls to Start() beyond the first without an intervening QAManagerBase.Stop() call are ignored.

See also

- "QAManagerBase.Stop method [QAnywhere .NET]" on page 265

Stop method

Stops the QAManagerBase's reception of incoming messages.

Visual Basic syntax

```
Public Sub Stop()
```

C# syntax

```
public void Stop()
```

Exceptions

- **QAException class** Thrown if there is a problem stopping the QAManagerBase instance.

Remarks

The messages are not lost. They will not be received until the manager is started again. Any calls to Stop() beyond the first without an intervening QAManagerBase.Start() call are ignored.

See also

- "QAManagerBase.Start method [QAnywhere .NET]" on page 264

TriggerSendReceive method

Causes a synchronization with the QAnywhere message server, uploading any messages addressed to other clients, and downloading any messages addressed to the local client.

Visual Basic syntax

```
Public Sub TriggerSendReceive()
```

C# syntax

```
public void TriggerSendReceive()
```

Exceptions

- **QAException class** Thrown if there is a problem triggering the send/receive.

Remarks

QAManagerBase TriggerSendReceive results in immediate message synchronization between a QAnywhere Agent and the central messaging server. A manual TriggerSendReceive call results in immediate message transmission, independent of the QAnywhere Agent transmission policies.

QAnywhere Agent transmission policies determine how message transmission occurs. For example, message transmission can occur automatically at regular intervals, when your client receives a push notification, or when you call the QAManagerBase.PutMessage method to send a message.

For more information, see "Determining when message transmission should occur on the client" on page 46.

See also

- "QAManagerBase.PutMessage method [QAnywhere .NET]" on page 251

Mode property

Returns the QAManager acknowledgement mode for received messages.

Visual Basic syntax

```
Public ReadOnly Property Mode As AcknowledgementMode
```

C# syntax

```
public AcknowledgementMode Mode {get;}
```

Remarks

For a list of possible values, see AcknowledgementMode.
AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT and AcknowledgementMode.IMPLICIT_ACKNOWLEDGEMENT apply to QAManager instances; AcknowledgementMode.TRANSACTIONAL is the mode for QATransactionalManager instances.

QAManagerFactory class

This class acts as a factory class for creating QATransactionalManager and QAManager objects.

Visual Basic syntax

```
Public MustInherit Class QAManagerFactory
```

C# syntax

```
public abstract class QAManagerFactory
```

Members

All members of QAManagerFactory class, including all inherited members.

Name	Description
CreateQAManager method	Returns a new QAManager instance with the specified properties.
CreateQATransactionalManager method	Returns a new QATransactionalManager instance with the specified properties.
Instance property	A singleton QAManagerFactory instance.

Remarks

You can only have one instance of QAManagerFactory.

CreateQAManager method

Returns a new QAManager instance with the specified properties.

Overload list

Name	Description
CreateQAManager() method	Returns a new QAManager instance with default properties.
CreateQAManager(Hashtable) method	Returns a new QAManager instance with the specified properties as a Hashtable.
CreateQAManager(string) method	Returns a new QAManager instance with the specified properties.

CreateQAManager() method

Returns a new QAManager instance with default properties.

Visual Basic syntax

```
Public Overridable Function CreateQAManager() As QAManager
```

C# syntax

```
public virtual abstract QAManager CreateQAManager()
```

Returns

A new QAManager instance.

Exceptions

- **QAException class** Thrown if there is a problem creating the manager.

Remarks

You can use the QAManagerBase.SetProperty to set QAnywhere manager configuration properties programmatically after you create the instance.

For a list of QAnywhere manager configuration properties, see "QAnywhere manager configuration properties" on page 80.

See also

- "QAManager interface [QAnywhere .NET]" on page 221

CreateQAManager(Hashtable) method

Returns a new QAManager instance with the specified properties as a Hashtable.

Visual Basic syntax

```
Public Overridable Function CreateQAManager(
     ByVal properties As Hashtable
) As QAManager
```

C# syntax

```
public virtual abstract QAManager CreateQAManager(Hashtable properties)
```

Parameters

- **properties** A hashtable for configuring the QAManager instance.

Returns

A new QAManager instance.

Exceptions

- **QAException class** Thrown if there is a problem creating the manager.

Remarks

If the hashtable parameter is null, the QAManager is created using default properties. You can use the QAManagerBase.SetProperty to set QAnywhere manager configuration properties programmatically after you create the instance.

For a list of QAnywhere manager configuration properties, see "QAnywhere manager configuration properties" on page 80.

See also

- "QAManager interface [QAnywhere .NET]" on page 221

CreateQAManager(string) method

Returns a new QAManager instance with the specified properties.

Visual Basic syntax

```
Public Overridable Function CreateQAManager(
     ByVal iniFile As String
) As QAManager
```

C# syntax

```
public virtual abstract QAManager CreateQAManager(string iniFile)
```

Parameters

- **iniFile** A properties file for configuring the QAManager instance.

Returns

A new QAManager instance.

Exceptions

- **QAException class** Thrown if there is a problem creating the manager.

Remarks

If the properties file parameter is null, the QAManager is created using default properties. You can use the QAManagerBase.SetProperty to set QAnywhere manager configuration properties programmatically after you create the instance.

For a list of QAnywhere manager configuration properties, see "QAnywhere manager configuration properties" on page 80.

For more information, see "QAnywhere manager configuration property settings in a file" on page 82.

See also

- "QAManager interface [QAnywhere .NET]" on page 221

CreateQATransactionalManager method

Returns a new QATransactionalManager instance with the specified properties.

Overload list

Name	Description
CreateQATransactionalManager() method	Returns a new QATransactionalManager instance with default properties.
CreateQATransactionalManager(Hashtable) method	Returns a new QATransactionalManager instance with the specified properties as a Hashtable.
CreateQATransactionalManager(string) method	Returns a new QATransactionalManager instance with the specified properties.

CreateQATransactionalManager() method

Returns a new QATransactionalManager instance with default properties.

Visual Basic syntax

```
Public Overridable Function CreateQATransactionalManager()
    As QATransactionalManager
```

C# syntax

```
public virtual abstract QATransactionalManager
CreateQATransactionalManager()
```

Returns

A new QATransactionalManager instance.

Exceptions

- **QAException class** Thrown if there is a problem creating the manager.

Remarks

You can use the QAManagerBase.SetProperty to set QAnywhere manager configuration properties programmatically after you create the instance.

For a list of QAnywhere manager configuration properties, see "QAnywhere manager configuration properties" on page 80.

See also

- "QATransactionalManager interface [QAnywhere .NET]" on page 295

CreateQATransactionalManager(Hashtable) method

Returns a new QATransactionalManager instance with the specified properties as a Hashtable.

Visual Basic syntax

```
Public Overridable Function CreateQATransactionalManager(
     ByVal properties As Hashtable
) As QATransactionalManager
```

C# syntax

```
public virtual abstract QATransactionalManager
CreateQATransactionalManager(
     Hashtable properties
)
```

Parameters

- **properties** A hashtable for configuring the QATransactionalManager instance.

Returns

A new QATransactionalManager

Exceptions

- **QAException class** Thrown if there is a problem creating the manager.

Remarks

If the hashtable parameter is null, the QATransactionalManager is created using default properties. You can use the QAManagerBase.SetProperty to set QAnywhere manager configuration properties programmatically after you create the instance.

For a list of QAnywhere manager configuration properties, see "QAnywhere manager configuration properties" on page 80.

See also

- "QATransactionalManager interface [QAnywhere .NET]" on page 295

CreateQATransactionalManager(string) method

Returns a new QATransactionalManager instance with the specified properties.

Visual Basic syntax

```
Public Overridable Function CreateQATransactionalManager(
     ByVal iniFile As String
) As QATransactionalManager
```

C# syntax

```
public virtual abstract QATransactionalManager
CreateQATransactionalManager(
     string iniFile
)
```

Parameters

- **iniFile** A properties file for configuring the QATransactionalManager instance, or null to create the QATransactionalManager instance with default properties.

Returns

A new QATransactionalManager

Exceptions

- **QAException class** Thrown if there is a problem creating the manager.

Remarks

If the properties file parameter is null, the QATransactionalManager is created using default properties. You can use the QAManagerBase.SetProperty to set QAnywhere Manager configuration properties programmatically after you create the instance.

For a list of QAnywhere Manager configuration properties, see "QAnywhere manager configuration properties" on page 80.

For more information, see "QAnywhere manager configuration property settings in a file" on page 82.

See also

- "QATransactionalManager interface [QAnywhere .NET]" on page 295

Instance property

A singleton QAManagerFactory instance.

Visual Basic syntax

```
Public Shared ReadOnly Property Instance As QAManagerFactory
```

C# syntax

```
public QAManagerFactory Instance {get;}
```

Exceptions

- **QAException class** Thrown if there is a problem creating the manager factory.

QAMessage interface

Provides an interface to set message properties and header fields.

Visual Basic syntax

```
Public Interface QAMessage
```

C# syntax

```
public interface QAMessage
```

Derived classes

- "QABinaryMessage interface [QAnywhere .NET]" on page 191
- "QATextMessage interface [QAnywhere .NET]" on page 291

Members

All members of QAMessage interface, including all inherited members.

Name	Description
ClearBody method	Clears the body of the message.
ClearProperties method	Clears all the properties of the message.
GetBooleanProperty method	Gets a boolean message property.
GetByteProperty method	Gets a byte message property.
GetDoubleProperty method	Gets a double message property.
GetFloatProperty method	Gets a float message property.
GetIntProperty method	Gets an int message property.

Name	Description
GetLongProperty method	Gets a long message property.
GetProperty method	Gets a message property.
GetPropertyNames method	Gets an enumerator over the property names of the message.
GetPropertyType method	Returns the property type of the given property.
GetSbyteProperty method	Gets a signed byte message property.
GetShortProperty method	Gets a short message property.
GetStringProperty method	Gets a string message property.
PropertyExists method	Indicates whether the given property has been set for this message.
SetBooleanProperty method	Sets a boolean property.
SetByteProperty method	Sets a byte property.
SetDoubleProperty method	Sets a double property.
SetFloatProperty method	Sets a float property.
SetIntProperty method	Sets an int property.
SetLongProperty method	Sets a long property.
SetProperty method	Sets a property.
SetSbyteProperty method	Sets a signed byte property.
SetShortProperty method	Sets a short property.
SetStringProperty method	Sets a string property.
Address property	The destination address for the QAMessage instance.
Expiration property	Gets the message's expiration value.
InReplyToID property	The message id of the message for which this message is a reply.
MessageID property	The globally unique message id of the message.
Priority property	The priority of the message (ranging from 0 to 9).
Redelivered property	Indicates whether the message has been previously received but not acknowledged.

Name	Description
ReplyToAddress property	The reply to address of this message.
Timestamp property	The message timestamp.

Remarks

The derived classes QABinaryMessage and QATextMessage provide specialized methods to read and write to the message body. You can use QAMessage methods to set predefined or custom message properties.

For a list of predefined property names, see the MessageProperties.

For more information about setting message properties and header fields, see "QAnywhere messages" on page 13.

See also

- "QABinaryMessage interface [QAnywhere .NET]" on page 191
- "QATextMessage interface [QAnywhere .NET]" on page 291

ClearBody method

Clears the body of the message.

Visual Basic syntax

```
Public Sub ClearBody()
```

C# syntax

```
public void ClearBody()
```

ClearProperties method

Clears all the properties of the message.

Visual Basic syntax

```
Public Sub ClearProperties()
```

C# syntax

```
public void ClearProperties()
```

GetBooleanProperty method

Gets a boolean message property.

Visual Basic syntax

```
Public Function GetBooleanProperty(ByVal propName As String) As Boolean
```

C# syntax

```
public bool GetBooleanProperty(string propName)
```

Parameters

- **propName** The property name.

Returns

The property value.

Exceptions

- **QAException class** Thrown if there is a conversion error getting the property value or if the property does not exist.

Remarks

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

See also

- "MessageProperties class [QAnywhere .NET]" on page 181

GetByteProperty method

Gets a byte message property.

Visual Basic syntax

```
Public Function GetByteProperty(ByVal propName As String) As Byte
```

C# syntax

```
public byte GetByteProperty(string propName)
```

Parameters

- **propName** The property name.

Returns

The property value.

Exceptions

- **QAException class** Thrown if there is a conversion error getting the property value or if the property does not exist.

Remarks

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

See also

- "MessageProperties class [QAnywhere .NET]" on page 181

GetDoubleProperty method

Gets a double message property.

Visual Basic syntax

```
Public Function GetDoubleProperty(ByVal propName As String) As Double
```

C# syntax

```
public double GetDoubleProperty(string propName)
```

Parameters

- **propName** The property name.

Returns

The property value.

Exceptions

- **QAException class** Thrown if there is a conversion error getting the property value or if the property does not exist.

Remarks

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

See also

- "MessageProperties class [QAnywhere .NET]" on page 181

GetFloatProperty method

Gets a float message property.

Visual Basic syntax

```
Public Function GetFloatProperty(ByVal propName As String) As Single
```

C# syntax

```
public float GetFloatProperty(string propName)
```


Parameters

- **propName** The property name.

Returns

The property value.

Exceptions

- **QAException class** Thrown if there is a conversion error getting the property value or if the property does not exist.

Remarks

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

See also

- "MessageProperties class [QAnywhere .NET]" on page 181

GetIntProperty method

Gets an int message property.

Visual Basic syntax

```
Public Function GetIntProperty(ByVal propName As String) As Integer
```

C# syntax

```
public int GetIntProperty(string propName)
```

Parameters

- **propName** The property name.

Returns

The property value.

Exceptions

- **QAException class** Thrown if there is a conversion error getting the property value or if the property does not exist.

Remarks

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

See also

- "MessageProperties class [QAnywhere .NET]" on page 181

GetLongProperty method

Gets a long message property.

Visual Basic syntax

```
Public Function GetLongProperty(ByVal propName As String) As Long
```

C# syntax

```
public long GetLongProperty(string propName)
```

Parameters

- **propName** The property name.

Returns

The property value.

Exceptions

- **QAException class** Thrown if there is a conversion error getting the property value or if the property does not exist.

Remarks

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

See also

- "MessageProperties class [QAnywhere .NET]" on page 181

GetProperty method

Gets a message property.

Visual Basic syntax

```
Public Function GetProperty(ByVal propName As String) As Object
```

C# syntax

```
public Object GetProperty(string propName)
```

Parameters

- **propName** The property name.

Returns

The property value.

Exceptions

- **QAException class** Thrown if the property does not exist.

Remarks

The property must be one of the acceptable simple types, string, or DateTime.

GetPropertyNames method

Gets an enumerator over the property names of the message.

Visual Basic syntax

```
Public Function GetPropertyNames() As System.Collections.IEnumerator
```

C# syntax

```
public System.Collections.IEnumerator GetPropertyNames()
```

Returns

An enumerator over the message property names.

GetPropertyType method

Returns the property type of the given property.

Visual Basic syntax

```
Public Function GetPropertyType(
     ByVal propName As String
) As PropertyType
```

C# syntax

```
public PropertyType GetPropertyType(string propName)
```

Parameters

- **propName** The name of the property.

Returns

The property type.

GetSbyteProperty method

Gets a signed byte message property.

Visual Basic syntax

```
Public Function GetSbyteProperty(ByVal propName As String) As SByte
```

C# syntax

```
public sbyte GetSbyteProperty(string propName)
```

Parameters

- **propName** the property name.

Returns

The property value.

Exceptions

- **QAException class** Thrown if there is a conversion error getting the property value or if the property does not exist.

Remarks

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

See also

- "MessageProperties class [QAnywhere .NET]" on page 181

GetShortProperty method

Gets a short message property.

Visual Basic syntax

```
Public Function GetShortProperty(ByVal propName As String) As Short
```

C# syntax

```
public short GetShortProperty(string propName)
```

Parameters

- **propName** The property name.

Returns

The property value.

Exceptions

- **QAException class** Thrown if there is a conversion error getting the property value or if the property does not exist.

Remarks

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

See also

- "MessageProperties class [QAnywhere .NET]" on page 181

GetStringProperty method

Gets a string message property.

Visual Basic syntax

```
Public Function GetStringProperty(ByVal propName As String) As String
```

C# syntax

```
public string GetStringProperty(string propName)
```

Parameters

- **propName** The property name.

Returns

The property value or null if the property does not exist.

Remarks

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

See also

- "MessageProperties class [QAnywhere .NET]" on page 181

PropertyExists method

Indicates whether the given property has been set for this message.

Visual Basic syntax

```
Public Function PropertyExists(ByVal propName As String) As Boolean
```

C# syntax

```
public bool PropertyExists(string propName)
```

Parameters

- **propName** The property name.

Returns

True if the property exists.

SetBooleanProperty method

Sets a boolean property.

Visual Basic syntax

```
Public Sub SetBooleanProperty(
     ByVal propName As String,
    ByVal val As Boolean
)
```

C# syntax

```
public void SetBooleanProperty(string propName, bool val)
```

Parameters

- **propName** The property name.
- **val** The property value.

Remarks

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

See also

- "MessageProperties class [QAnywhere .NET]" on page 181

SetByteProperty method

Sets a byte property.

Visual Basic syntax

```
Public Sub SetByteProperty(ByVal propName As String, ByVal val As Byte)
```

C# syntax

```
public void SetByteProperty(string propName, byte val)
```

Parameters

- **propName** The property name.
- **val** The property value.

Remarks

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

See also

- "MessageProperties class [QAnywhere .NET]" on page 181

SetDoubleProperty method

Sets a double property.

Visual Basic syntax

```
Public Sub SetDoubleProperty(
     ByVal propName As String,
    ByVal val As Double
)
```

C# syntax

```
public void SetDoubleProperty(string propName, double val)
```

Parameters

- **propName** The property name.
- **val** The property value.

Remarks

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

See also

- "MessageProperties class [QAnywhere .NET]" on page 181

SetFloatProperty method

Sets a float property.

Visual Basic syntax

```
Public Sub SetFloatProperty(
     ByVal propName As String,
    ByVal val As Single
)
```

C# syntax

```
public void SetFloatProperty(string propName, float val)
```

Parameters

- **propName** The property name.
- **val** The property value.

Remarks

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

See also

- "MessageProperties class [QAnywhere .NET]" on page 181

SetIntProperty method

Sets an int property.

Visual Basic syntax

```
Public Sub SetIntProperty(
     ByVal propName As String,
    ByVal val As Integer
)
```

C# syntax

```
public void SetIntProperty(string propName, int val)
```

Parameters

- **propName** The property name.
- **val** The property value.

Remarks

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

See also

- "MessageProperties class [QAnywhere .NET]" on page 181

SetLongProperty method

Sets a long property.

Visual Basic syntax

```
Public Sub SetLongProperty(ByVal propName As String, ByVal val As Long)
```


C# syntax

```
public void SetLongProperty(string propName, long val)
```

Parameters

- **propName** The property name.
- **val** The property value.

Remarks

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

See also

- "MessageProperties class [QAnywhere .NET]" on page 181

SetProperty method

Sets a property.

Visual Basic syntax

```
Public Sub SetProperty(ByVal propName As String, ByVal val As Object)
```

C# syntax

```
public void SetProperty(string propName, Object val)
```

Parameters

- **propName** The property name.
- **val** The property value.

Remarks

The property type must be one of the acceptable simple types, string, or DateTime.

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

See also

- "MessageProperties class [QAnywhere .NET]" on page 181

SetSbyteProperty method

Sets a signed byte property.

Visual Basic syntax

```
Public Sub SetSbyteProperty(
     ByVal propName As String,
    ByVal val As SByte
)
```

C# syntax

```
public void SetSbyteProperty(string propName, sbyte val)
```

Parameters

- **propName** The property name.
- **val** The property value.

Remarks

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

See also

- "MessageProperties class [QAnywhere .NET]" on page 181

SetShortProperty method

Sets a short property.

Visual Basic syntax

```
Public Sub SetShortProperty(
     ByVal propName As String,
    ByVal val As Short
)
```

C# syntax

```
public void SetShortProperty(string propName, short val)
```

Parameters

- **propName** The property name.
- **val** The property value.

Remarks

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

See also

- "MessageProperties class [QAnywhere .NET]" on page 181

SetStringProperty method

Sets a string property.

Visual Basic syntax

```
Public Sub SetStringProperty(
     ByVal propName As String,
    ByVal val As String
)
```

C# syntax

```
public void SetStringProperty(string propName, string val)
```

Parameters

- **propName** The property name.
- **val** The property value.

Remarks

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

See also

- "MessageProperties class [QAnywhere .NET]" on page 181

Address property

The destination address for the QAMessage instance.

Visual Basic syntax

```
Public ReadOnly Property Address As String
```

C# syntax

```
public string Address {get;}
```

Remarks

When a message is sent, this field is ignored. After completion of a send operation, the field holds the destination address specified in QAManagerBase.PutMessage.

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

See also

- "QAManagerBase.PutMessage method [QAnywhere .NET]" on page 251

Expiration property

Gets the message's expiration value.

Visual Basic syntax

```
Public ReadOnly Property Expiration As Date
```

C# syntax

```
public DateTime Expiration {get;}
```

Remarks

When a message is sent, the Expiration header field is left unassigned. After completion of the send method, it holds the expiration time of the message.

This is a read-only property because the expiration time of a message is set by adding the time-to-live argument of QAManagerBase.PutMessageTimeToLive to the current time.

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

InReplyToID property

The message id of the message for which this message is a reply.

Visual Basic syntax

```
Public Property InReplyToID As String
```

C# syntax

```
public string InReplyToID {get;set;}
```

Remarks

May be null.

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

MessageID property

The globally unique message id of the message.

Visual Basic syntax

```
Public ReadOnly Property MessageID As String
```


C# syntax

```
public string MessageID {get;}
```

Remarks

This property is null until a message is put.

When a message is sent using QAManagerBase.PutMessage, the MessageID is null and can be ignored. When the send method returns, it contains an assigned value.

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

See also

- "QAManagerBase.PutMessage method [QAnywhere .NET]" on page 251

Priority property

The priority of the message (ranging from 0 to 9).

Visual Basic syntax

```
Public Property Priority As Integer
```

C# syntax

```
public int Priority {get;set;}
```

Remarks

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

Redelivered property

Indicates whether the message has been previously received but not acknowledged.

Visual Basic syntax

```
Public ReadOnly Property Redelivered As Boolean
```

C# syntax

```
public bool Redelivered {get;}
```

Remarks

Redelivered is set by a receiving QAManager when it detects that a message being received was received before.

For example, an application receives a message using a QAManager opened with AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT, and shuts down without acknowledging the message. When the application starts again and receives the same message the Redelivered header will be true.

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

See also

- "QAManager interface [QAnywhere .NET]" on page 221
- "AcknowledgementMode enumeration [QAnywhere .NET]" on page 302

ReplyToAddress property

The reply to address of this message.

Visual Basic syntax

```
Public Property ReplyToAddress As String
```

C# syntax

```
public string ReplyToAddress {get;set;}
```

Remarks

May be null.

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

Timestamp property

The message timestamp.

Visual Basic syntax

```
Public ReadOnly Property Timestamp As Date
```

C# syntax

```
public DateTime Timestamp {get;}
```

Remarks

This Timestamp header field contains the time a message was created.

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

QATextMessage interface

QATextMessage inherits from the QAMessage class and adds a text message body.

Visual Basic syntax

```
Public Interface QATextMessage Inherits QAMessage
```

C# syntax

```
public interface QATextMessage : QAMessage
```

Base classes

- "QAMessage interface [QAnywhere .NET]" on page 272

Members

All members of QATextMessage interface, including all inherited members.

Name	Description
ClearBody method	Clears the body of the message.
ClearProperties method	Clears all the properties of the message.
GetBooleanProperty method	Gets a boolean message property.
GetByteProperty method	Gets a byte message property.
GetDoubleProperty method	Gets a double message property.
GetFloatProperty method	Gets a float message property.
GetIntProperty method	Gets an int message property.
GetLongProperty method	Gets a long message property.
GetProperty method	Gets a message property.
GetPropertyNames method	Gets an enumerator over the property names of the message.
GetPropertyType method	Returns the property type of the given property.
GetSbyteProperty method	Gets a signed byte message property.
GetShortProperty method	Gets a short message property.
GetStringProperty method	Gets a string message property.
PropertyExists method	Indicates whether the given property has been set for this message.

Name	Description
ReadText method	Read unread text into the given buffer.
Reset method	Resets the text position of the message to the beginning.
SetBooleanProperty method	Sets a boolean property.
SetByteProperty method	Sets a byte property.
SetDoubleProperty method	Sets a double property.
SetFloatProperty method	Sets a float property.
SetIntProperty method	Sets an int property.
SetLongProperty method	Sets a long property.
SetProperty method	Sets a property.
SetSbyteProperty method	Sets a signed byte property.
SetShortProperty method	Sets a short property.
SetStringProperty method	Sets a string property.
WriteText method	Append text to the text of the message.
Address property	The destination address for the QAMessage instance.
Expiration property	Gets the message's expiration value.
InReplyToID property	The message id of the message for which this message is a reply.
MessageID property	The globally unique message id of the message.
Priority property	The priority of the message (ranging from 0 to 9).
Redelivered property	Indicates whether the message has been previously received but not acknowledged.
ReplyToAddress property	The reply to address of this message.
Text property	The message text.
TextLength property	The length, in characters, of the message.
Timestamp property	The message timestamp.

Remarks

QATextMessage provides methods to read from and write to the text message body.

When the message is first created, the body of the message is in write-only mode. After a message has been sent, the client that sent it can retain and modify it without affecting the message that has been sent. The same message object can be sent multiple times.

When a message is received, the provider has called QATextMessage.Reset() so that the message body is in read-only mode and reading of values starts from the beginning of the message body.

See also

- "QABinaryMessage interface [QAnywhere .NET]" on page 191
- "QAMessage interface [QAnywhere .NET]" on page 272

ReadText method

Read unread text into the given buffer.

Visual Basic syntax

```
Public Function ReadText(ByVal buf As StringBuilder) As Integer
```

C# syntax

```
public int ReadText(StringBuilder buf)
```

Parameters

- **buf** Target buffer for any read text.

Returns

The number of characters read or -1 if there are no more characters to read.

Remarks

Any additional unread text must be read by subsequent calls to this method. Text is read from the beginning of any unread text.

Reset method

Resets the text position of the message to the beginning.

Visual Basic syntax

```
Public Sub Reset()
```

C# syntax

```
public void Reset()
```

WriteText method

Append text to the text of the message.

Visual Basic syntax

```
Public Sub WriteText(ByVal val As String)
```

C# syntax

```
public void WriteText(string val)
```

Parameters

- **val** The text to append.

Text property

The message text.

Visual Basic syntax

```
Public Property Text As String
```

C# syntax

```
public string Text {get;set;}
```

Remarks

If the message exceeds the maximum size specified by the QAManager.MAX_IN_MEMORY_MESSAGE_SIZE, this property is null. In this case, use the QATextMessage.ReadText method to read the text.

For more information about QAManager properties, see "QAnywhere manager configuration properties" on page 80.

See also

- "QATextMessage.ReadText method [QAnywhere .NET]" on page 293

TextLength property

The length, in characters, of the message.

Visual Basic syntax

```
Public ReadOnly Property TextLength As Long
```

C# syntax

```
public long TextLength {get;}
```


QATransactionalManager interface

The QATransactionalManager class derives from QAManagerBase and manages transactional QAnywhere messaging operations.

Visual Basic syntax

```
Public Interface QATransactionalManager Inherits QAManagerBase
```

C# syntax

```
public interface QATransactionalManager : QAManagerBase
```

Base classes

- "QAManagerBase interface [QAnywhere .NET]" on page 228

Members

All members of QATransactionalManager interface, including all inherited members.

Name	Description
BrowseMessages method	This method is deprecated.
BrowseMessagesByID method	Browses the message with the given message ID.
BrowseMessagesByQueue method	Browses the next available messages waiting that have been sent to the given address.
BrowseMessagesBySelector method	Browses messages queued in the message store that satisfy the given selector.
CancelMessage method	Cancels the message with the given message ID.
Close method	Closes the connection to the QAnywhere message system and releases any resources used by the QAManagerBase.
Commit method	Commits the current transaction and begins a new transaction.
CreateBinaryMessage method	Creates a QABinaryMessage object.
CreateTextMessage method	Creates a QATextMessage object.
GetBooleanStoreProperty method	Gets a boolean value for a predefined or custom message store property.
GetDoubleStoreProperty method	Gets a double value for a predefined or custom message store property.

Name	Description
GetFloatStoreProperty method	Gets a float value for a predefined or custom message store property.
GetIntStoreProperty method	Gets an int value for a predefined or custom message store property.
GetLongStoreProperty method	Gets a long value for a predefined or custom message store property.
GetMessage method	Returns the next available QAMessage sent to the specified address.
GetMessageBySelector method	Returns the next available QAMessage sent to the specified address that satisfies the given selector.
GetMessageBySelectorNoWait method	Returns the next available QAMessage sent to the given address that satisfies the given selector.
GetMessageBySelectorTimeout method	Returns the next available QAMessage sent to the given address that satisfies the given selector.
GetMessageNoWait method	Returns the next available QAMessage sent to the given address.
GetMessageTimeout method	Returns the next available QAMessage sent to the given address.
GetQueueDepth method	Returns the total depth of all queues, based on a given filter.
GetSbyteStoreProperty method	Gets a signed byte value for a predefined or custom message store property.
GetShortStoreProperty method	Gets a short value for a predefined or custom message store property.
GetStoreProperty method	Gets a System.Object representing a message store property.
GetStorePropertyNames method	Gets an enumerator over the message store property names.
GetStringStoreProperty method	Gets a string value for a predefined or custom message store property.
Open method	Opens a QATransactionalManager instance.
PropertyExists method	Tests if there currently exists a value for the given the property.
PutMessage method	Prepares a message to send to another QAnywhere client.
PutMessageTimeToLive method	Prepares a message to send to another QAnywhere client.

Name	Description
ReOpen method	Reopens the QAManagerBase.
Rollback method	Rolls back the current transaction and begins a new transaction.
SetBooleanStoreProperty method	Sets a predefined or custom message store property to a boolean value.
SetDoubleStoreProperty method	Sets a predefined or custom message store property to a double value.
SetExceptionListener method	Sets an ExceptionListener delegate to receive QAExceptions when processing QAnywhere messages asynchronously.
SetExceptionListener2 method	Sets an ExceptionListener2 delegate to receive QAExceptions when processing QAnywhere messages asynchronously.
SetFloatStoreProperty method	Sets a predefined or custom message store property to a float value.
SetIntStoreProperty method	Sets a predefined or custom message store property to an int value.
SetLongStoreProperty method	Sets a predefined or custom message store property to a long value.
SetMessageListener method	Sets a MessageListener delegate to receive QAnywhere messages asynchronously.
SetMessageListener2 method	Sets a MessageListener2 delegate to receive QAnywhere messages asynchronously.
SetMessageListenerBySelector method	Sets a MessageListener delegate to receive QAnywhere messages asynchronously, with a message selector.
SetMessageListenerBySelector2 method	Sets a MessageListener2 delegate to receive QAnywhere messages asynchronously, with a message selector.
SetProperty method	Allows you to set QAnywhere Manager configuration properties programmatically.
SetSbyteStoreProperty method	Sets a predefined or custom message store property to an sbyte (signed byte) value.
SetShortStoreProperty method	Sets a predefined or custom message store property to a short value.
SetStoreProperty method	Sets a predefined or custom message store property to a System.Object value.

Name	Description
SetStringStoreProperty method	Sets a predefined or custom message store property to a string value.
Start method	Starts the QAManagerBase for receiving incoming messages in message listeners.
Stop method	Stops the QAManagerBase's reception of incoming messages.
TriggerSendReceive method	Causes a synchronization with the QAnywhere message server, uploading any messages addressed to other clients, and downloading any messages addressed to the local client.
Mode property	Returns the QAManager acknowledgement mode for received messages.

Remarks

For a detailed description of derived behavior, see QAManagerBase.

The QATransactionalManager can only be used for transactional acknowledgement. Use the QATransactionalManager.Commit() method to commit all QAManagerBase.PutMessage and QAManagerBase.GetMessage invocations.

For more information, see "Transactional messaging implementation" on page 63.

See also

- "QATransactionalManager interface [QAnywhere .NET]" on page 295

Commit method

Commits the current transaction and begins a new transaction.

Visual Basic syntax

```
Public Sub Commit()
```

C# syntax

```
public void Commit()
```

Exceptions

- **QAException class** Thrown if there is a problem committing.

Remarks

> **Note**
> This method commits all QAManagerBase.PutMessage and QAManagerBase.GetMessage invocations. The first transaction begins with the call to QATransactionalManager.Open().

See also

- "QATransactionalManager interface [QAnywhere .NET]" on page 295

Open method

Opens a QATransactionalManager instance.

Visual Basic syntax

```
Public Sub Open()
```

C# syntax

```
public void Open()
```

Exceptions

- **QAException class** Thrown if there is a problem opening the manager

Remarks

The Open method must be the first method called after creating a manager.

See also

- "QATransactionalManager interface [QAnywhere .NET]" on page 295

Rollback method

Rolls back the current transaction and begins a new transaction.

Visual Basic syntax

```
Public Sub Rollback()
```

C# syntax

```
public void Rollback()
```

Exceptions

- **QAException class** Thrown if there is a problem rolling back

Remarks

This method rolls back all uncommitted QAManagerBase.PutMessage and QAManagerBase.GetMessage invocations.

See also

- "QATransactionalManager interface [QAnywhere .NET]" on page 295

ExceptionListener delegate

ExceptionListener delegate definition.

Visual Basic syntax

```
Public Delegate Sub ExceptionListener(
     ByVal ex As QAException,
    ByVal msg As QAMessage
)
```

C# syntax

```
public delegate void ExceptionListener(QAException ex, QAMessage msg);
```

Parameters

- **ex** The exception that occurred.
- **msg** The message that was received, or null if the message could not be constructed.

Remarks

You pass an ExceptionListener to the QAManagerBase.SetExceptionListener.

ExceptionListener2 delegate

ExceptionListener2 delegate definition.

Visual Basic syntax

```
Public Delegate Sub ExceptionListener2(
     ByVal mgr As QAManagerBase,
    ByVal ex As QAException,
    ByVal msg As QAMessage
)
```

C# syntax

```
public delegate void ExceptionListener2(
     QAManagerBase mgr,
    QAException ex,
    QAMessage msg
);
```


Parameters

- **mgr** The QAManagerBase that processed the message.
- **ex** The exception that occurred.
- **msg** The message that was received, or null if the message could not be constructed.

Remarks

You pass an ExceptionListener2 to the QAManagerBase.SetExceptionListener2.

MessageListener delegate

MessageListener delegate definition.

Visual Basic syntax

```
Public Delegate Sub MessageListener(ByVal msg As QAMessage)
```

C# syntax

```
public delegate void MessageListener(QAMessage msg);
```

Parameters

- **msg** The message that was received.

Remarks

You pass a MessageListener to the QAManagerBase.SetMessageListener method.

See also

- "QAManagerBase.SetMessageListener method [QAnywhere .NET]" on page 257

MessageListener2 delegate

MessageListener2 delegate definition.

Visual Basic syntax

```
Public Delegate Sub MessageListener2(
     ByVal mgr As QAManagerBase,
    ByVal msg As QAMessage
)
```

C# syntax

```
public delegate void MessageListener2(QAManagerBase mgr, QAMessage msg);
```

Parameters

- **mgr** The QAManagerBase that received the message.

- **msg** The message that was received.

Remarks

You pass a MessageListener2 to the QAManagerBase.SetMessageListener2.

AcknowledgementMode enumeration

Indicates how messages should be acknowledged by QAnywhere client applications.

Visual Basic syntax

```
Public Enum AcknowledgementMode
```

C# syntax

```
public enum AcknowledgementMode
```

Members

Member name	Description
EXPLICIT_ACKNOWLEDGEMENT	Indicates that received messages are acknowledged using one of the QAManager acknowledge methods.
IMPLICIT_ACKNOWLEDGEMENT	Indicates that all messages are acknowledged as soon as they are received by a client application. If you receive messages synchronously, messages are acknowledged as soon as the QAManagerBase.GetMessage(string) method returns. If you receive messages asynchronously, the message is acknowledged as soon as the event handling function returns.
TRANSACTIONAL	This mode indicates that messages are only acknowledged as part of the on going transaction. This mode is automatically assigned to QATransactionalManager instances.

Remarks

The implicit and explicit acknowledgement modes are assigned to a QAManager instance using the QAManager.Open(AcknowledgementMode) method.

For more information, see "Initializing a QAnywhere API" on page 52.

With implicit acknowledgement, messages are acknowledged as soon as they are received by a client application. With explicit acknowledgement, you must call one of the QAManager acknowledgement methods. The server propagates all status changes from client to client.

For more information, see "Receiving messages synchronously" on page 68 and "Receiving messages asynchronously" on page 69.

See also

- "QAManager interface [QAnywhere .NET]" on page 221
- "QATransactionalManager interface [QAnywhere .NET]" on page 295
- "QAManagerBase interface [QAnywhere .NET]" on page 228

MessageType enumeration

Defines constant values for the MessageProperties.MSG_TYPE message property.

Visual Basic syntax

```
Public Enum MessageType
```

C# syntax

```
public enum MessageType
```

Members

Member name	Description	Value
NETWORK_STATUS_NOTIFICATION	Identifies a QAnywhere system message used to notify QAnywhere client applications of network status changes. Network status changes apply to the device receiving the system message. Use MessageProperties.ADAPTERS, MessageProperties.RASNAMES, and MessageProperties.NETWORK_STATUS fields to identify new network status information. For more information, see "System queue" on page 58.	14
PUSH_NOTIFICATION	Identifies a QAnywhere system message used to notify QAnywhere client applications of push notifications. If you use the on-demand qaagent policy, a typical response is to call QAManagerBase.TriggerSendReceive() to receive messages waiting with the central message server. For more information, see "System queue" on page 58.	13
REGULAR	If no message type property exists then the message type is assumed to be REGULAR. This type of message is not treated specially by the message system.	0

See also

- "QAManagerBase.TriggerSendReceive method [QAnywhere .NET]" on page 265
- "MessageProperties class [QAnywhere .NET]" on page 181

Example

The following example shows the onSystemMessage method which is used to handle QAnywhere system messages. The message type is compared to MessageType.NETWORK_STATUS_NOTIFICATION.

```
private void onSystemMessage(QAMessage msg) {
    QATextMessage   t_msg;
    MessageType     msg_type;
    String          network_adapters;
    String          network_names;
    String          network_info;

    t_msg = (QATextMessage)msg;
    if(t_msg != null) {
          // Evaluate message type.
          msg_type =
(MessageType)t_msg.GetIntProperty( MessageProperties.MSG_TYPE );
        if(msg_type == MessageType.NETWORK_STATUS_NOTIFICATION) {
            // Handle network status notification...
        }
    }
}
```

PropertyType enumeration

QAMessage property type enumeration, corresponding naturally to the C# types.

Visual Basic syntax

```
Public Enum PropertyType
```

C# syntax

```
public enum PropertyType
```

Members

Member name	Description
BOOLEAN	Indicates a boolean property.
BYTE	Indicates a signed byte property.
DOUBLE	Indicates a double property.
FLOAT	Indicates a float property.
INT	Indicates an int property.

Member name	Description
LONG	Indicates an long property.
SHORT	Indicates a short property.
STRING	Indicates a string property.
UNKNOWN	Indicates an unknown property type, usually because the property is unknown.

QueueDepthFilter enumeration

Provides queue depth filter values for QAManagerBase.GetQueueDepth(QueueDepthFilter) and QAManagerBase.GetQueueDepth(string,QueueDepthFilter).

Visual Basic syntax

```
Public Enum QueueDepthFilter
```

C# syntax

```
public enum QueueDepthFilter
```

Members

Member name	Description
ALL	Count both incoming and outgoing messages. System messages and expired messages are not included in any queue depth counts.
INCOMING	Count only incoming messages. An incoming message is defined as a message whose originator is different than the agent ID of the message store.
LOCAL	Count only local messages in the message store. A local message is defined as a message whose originator and target are the agent ID of the message store.
OUTGOING	Count only outgoing messages. An outgoing message is defined as a message whose originator is the agent ID of the message store, and whose destination is not the agent ID of the message store.

See also

- "QAManagerBase.GetQueueDepth method [QAnywhere .NET]" on page 246

StatusCodes enumeration

This enumeration defines a set of codes for the status of a message.

Visual Basic syntax

```
Public Enum StatusCodes
```

C# syntax

```
public enum StatusCodes
```

Members

Member name	Description	Value
CANCELED	The message has been canceled. This code applies to the MessageProperties.STATUS.	40
EXPIRED	The message has expired because it was not received before its expiration time had passed. This code applies to the MessageProperties.STATUS.	30
FINAL	This constant is used to determine if a message has achieved a final state. A message has achieved a final state if and only if its status is greater than this constant. This code applies to the MessageProperties.STATUS.	20
LOCAL	The message is addressed to the local message store and will not be transmitted to the server. This code applies to the MessageProperties.TRANSMISSION_STATUS.	2
PENDING	The message has been sent but not received and acknowledged. This code applies to the MessageProperties.STATUS.	1
RECEIVED	The message has been received and acknowledged by the receiver. This code applies to the MessageProperties.STATUS.	60

Member name	Description	Value
RECEIVING	The message is in the process of being received, or it was received but not acknowledged. This code applies to the MessageProperties.STATUS.	10
TRANSMIT-TED	The message has been transmitted to the server. This code applies to the MessageProperties.TRANSMISSION_STATUS.	1
TRANSMIT-TING	The message is in the process of being transmitted to the server. This code applies to the MessageProperties.TRANSMISSION_STATUS.	3
UNRECEIV-ABLE	The message has been marked as unreceivable. The message is either malformed, or there were too many failed attempts to deliver it. This code applies to the MessageProperties.STATUS.	50
UNTRANS-MITTED	The message has not been transmitted to the server. This code applies to the MessageProperties.TRANSMISSION_STATUS.	0

QAnywhere .NET API reference for web services

Namespace

```
iAnywhere.QAnywhere.WS
```

WSBase class

This is the base class for the main web service proxy class generated by the mobile web service compiler.

Visual Basic syntax

```
Public Class WSBase
```

C# syntax

```
public class WSBase
```

Members

All members of WSBase class, including all inherited members.

Name	Description
WSBase constructor	Constructs a WSBase instance with the properties specified by a configuration property file.
ClearRequestProperties method	Clears all request properties that have been set for this WSBase.
GetResult method	Gets a WSResult object that represents the results of a web service request.
GetServiceID method	Gets the service ID for this instance of WSBase.
SetListener method	Sets a listener for the results of a given web service request.
SetProperty method	Sets a configuration property for this instance of WSBase.
SetQAManager method	Sets the QAManagerBase that is used by this web service client to do web service requests.
SetRequestProperty method	Sets a request property for webservice requests made by this WSBase.
SetServiceID method	Sets a user-defined ID for this instance of WSBase.

WSBase constructor

Constructs a WSBase instance with the properties specified by a configuration property file.

Overload list

Name	Description
WSBase() constructor	Constructs a WSBase instance with default properties.
WSBase(string) constructor	Constructs a WSBase instance with the properties specified by a configuration property file.

WSBase() constructor

Constructs a WSBase instance with default properties.

Visual Basic syntax

```
Public Sub New()
```

C# syntax

```
public WSBase()
```


Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem constructing the WSBase.

WSBase(string) constructor

Constructs a WSBase instance with the properties specified by a configuration property file.

Visual Basic syntax

```
Public Sub New(ByVal iniFile As String)
```

C# syntax

```
public WSBase(string iniFile)
```

Parameters

- **iniFile** A file containing configuration properties.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem constructing the WSBase.

Remarks

Valid configuration properties are:

LOG_FILE a file to which to log runtime information.

LOG_LEVEL a value between 0 and 6 that controls the verbosity of information logged, with 6 being the highest verbosity.

WS_CONNECTOR_ADDRESS the address of the web service connector in the MobiLink server.

The default WS_CONNECTOR_ADDRESS is ianywhere.connector.webservices\\.

ClearRequestProperties method

Clears all request properties that have been set for this WSBase.

Visual Basic syntax

```
Public Sub ClearRequestProperties()
```

C# syntax

```
public void ClearRequestProperties()
```

GetResult method

Gets a WSResult object that represents the results of a web service request.

Visual Basic syntax

```
Public Function GetResult(ByVal requestID As String) As WSResult
```

C# syntax

```
public WSResult GetResult(String requestID)
```

Parameters

- **requestID** The ID of the web service request.

Returns

A WSResult instance representing the results of the web service request.

See also

- "WSStatus enumeration [QAnywhere .NET]" on page 354

GetServiceID method

Gets the service ID for this instance of WSBase.

Visual Basic syntax

```
Public Function GetServiceID() As String
```

C# syntax

```
public String GetServiceID()
```

Returns

The service ID.

SetListener method

Sets a listener for the results of a given web service request.

Overload list

Name	Description
SetListener(string, WSListener) method	Sets a listener for the results of a given web service request.

Name	Description
SetListener(WSListener) method	Sets a listener for the results of all web service requests made by this instance of WSBase.

SetListener(string, WSListener) method

Sets a listener for the results of a given web service request.

Visual Basic syntax

```
Public Sub SetListener(
     ByVal requestID As String,
    ByVal listener As WSListener
)
```

C# syntax

```
public void SetListener(string requestID, WSListener listener)
```

Parameters

- **requestID** The ID of the web service request to which to listen for results.
- **listener** The listener object that gets called when the result of the given web service request is available.

Remarks

Listeners are typically used to get results of the asyncXYZ methods of the service.

To remove a listener, call SetListener with null as the listener.

Note
This method replaces the listener set by any previous call to SetListener.

SetListener(WSListener) method

Sets a listener for the results of all web service requests made by this instance of WSBase.

Visual Basic syntax

```
Public Sub SetListener(ByVal listener As WSListener)
```

C# syntax

```
public void SetListener(WSListener listener)
```

Parameters

- **listener** The listener object that gets called when the result of a web service request is available.

Remarks

Listeners are typically used to get results of the asyncXYZ methods of the service.

To remove a listener, call SetListener with null as the listener.

> **Note**
> This method replaces the listener set by any previous call to SetListener.

SetProperty method

Sets a configuration property for this instance of WSBase.

Visual Basic syntax

```
Public Sub SetProperty(ByVal property As String, ByVal val As String)
```

C# syntax

```
public void SetProperty(string property, string val)
```

Parameters

- **property** The property name to set.
- **val** The property value.

Remarks

Configuration properties must be set before any asynchronous or synchronous web service request is made. This method has no effect if it is called after a web service request has been made.

Valid configuration properties are:

LOG_FILE a file to which to log runtime information.

LOG_LEVEL a value between 0 and 6 that controls the verbosity of information logged, with 6 being the highest verbosity.

WS_CONNECTOR_ADDRESS the address of the web service connector in the MobiLink server. The default is: ianywhere.connector.webservices\\.

SetQAManager method

Sets the QAManagerBase that is used by this web service client to do web service requests.

Visual Basic syntax

```
Public Sub SetQAManager(ByVal mgr As QAManagerBase)
```


C# syntax

```
public void SetQAManager(QAManagerBase mgr)
```

Parameters

- **mgr** The QAManagerBase to use.

Remarks

> **Note**
> If you use an EXPLICIT_ACKNOWLEDGEMENT QAManager, you can acknowledge the result of an asynchronous web service request by calling the acknowledge() method of WSResult. The result of a synchronous web service request is automatically acknowledged, even in the case of an EXPLICIT_ACKNOWLEDGEMENT QAManager. If you use an IMPLICIT_ACKNOWLEDGEMENT QAManager, the result of any web service request is acknowledged automatically.

SetRequestProperty method

Sets a request property for webservice requests made by this WSBase.

Visual Basic syntax

```
Public Sub SetRequestProperty(
     ByVal name As String,
    ByVal value As Object
)
```

C# syntax

```
public void SetRequestProperty(string name, Object value)
```

Parameters

- **name** The property name to set.
- **value** The property value.

Remarks

A request property is set on each QAMessage that is sent by this WSBase, until the property is cleared. A request property is cleared by setting it to a null value. The type of the message property is determined by the class of the value parameter. For example, if value is an instance of Int32, then SetIntProperty is used to set the property on the QAMessage.

SetServiceID method

Sets a user-defined ID for this instance of WSBase.

Visual Basic syntax

```
Public Sub SetServiceID(ByVal serviceID As String)
```

C# syntax

```
public void SetServiceID(String serviceID)
```

Parameters

- **serviceID** The service ID.

Remarks

The service ID should be set to a value unique to this instance of WSBase. It is used internally to form a queue name for sending and receiving web service requests. Therefore, the service ID should be persisted between application sessions, in order to retrieve results of web service requests made in a previous session.

WSException class

This class represents an exception that occurred during processing of a web service request.

Visual Basic syntax

```
Public Class WSException Inherits System.Exception
```

C# syntax

```
public class WSException : System.Exception
```

Base classes

- System.Exception

Derived classes

- "WSFaultException class [QAnywhere .NET]" on page 318

Members

All members of WSException class, including all inherited members.

Name	Description
WSException constructor	Constructs a new exception with the specified error message.
GetBaseException method (Inherited from System.Exception)	When overridden in a derived class, returns the System.Exception that is the root cause of one or more subsequent exceptions.
GetObjectData method (Inherited from System.Exception)	When overridden in a derived class, sets the System.Runtime.Serialization.SerializationInfo with information about the exception.

Name	Description
GetType method (Inherited from System.Exception)	Gets the runtime type of the current instance.
ToString method (Inherited from System.Exception)	Creates and returns a string representation of the current exception.
Data property (Inherited from System.Exception)	Gets a collection of key/value pairs that provide additional user-defined information about the exception.
ErrorCode property	The error code associated with this exception.
HelpLink property (Inherited from System.Exception)	Gets or sets a link to the help file associated with this exception.
HResult property (Inherited from System.Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.
InnerException property (Inherited from System.Exception)	Gets the System.Exception instance that caused the current exception.
Message property (Inherited from System.Exception)	Gets a message that describes the current exception.
Source property (Inherited from System.Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace property (Inherited from System.Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite property (Inherited from System.Exception)	Gets the method that throws the current exception.
WS_STATUS_HTTP_ERROR field	Error code indicating that there was an error in the web service HTTP request made by the web services connector.
WS_STATUS_HTTP_OK field	Error code indicating that the webservice HTTP request by the web services connector was successful.
WS_STATUS_HTTP_RETRIES_EXCEEDED field	Error code indicating that the number of HTTP retries was exceeded the web services connector.
WS_STATUS_SOAP_PARSE_ERROR field	Error code indicating that there was an error in the web services runtime or in the webservices connector in parsing a SOAP response or request.

WSException constructor

Constructs a new exception with the specified error message.

Overload list

Name	Description
WSException(Exception) constructor	Constructs a new exception.
WSException(string) constructor	Constructs a new exception with the specified error message.
WSException(string, int) constructor	Constructs a new exception with the specified error message and error code.

WSException(Exception) constructor

Constructs a new exception.

Visual Basic syntax

```
Public Sub New(ByVal ex As Exception)
```

C# syntax

```
public WSException(Exception ex)
```

Parameters

- **ex** The exception.

WSException(string) constructor

Constructs a new exception with the specified error message.

Visual Basic syntax

```
Public Sub New(ByVal msg As String)
```

C# syntax

```
public WSException(string msg)
```

Parameters

- **msg** The error message.

WSException(string, int) constructor

Constructs a new exception with the specified error message and error code.

Visual Basic syntax

```
Public Sub New(ByVal msg As String, ByVal errorCode As Integer)
```

C# syntax

```
public WSException(string msg, int errorCode)
```

Parameters

- **msg** The error message.
- **errorCode** The error code.

ErrorCode property

The error code associated with this exception.

Visual Basic syntax

```
Public Property ErrorCode As Integer
```

C# syntax

```
public int ErrorCode {get;set;}
```

WS_STATUS_HTTP_ERROR field

Error code indicating that there was an error in the web service HTTP request made by the web services connector.

Visual Basic syntax

```
Public Shared WS_STATUS_HTTP_ERROR As Integer
```

C# syntax

```
public static int WS_STATUS_HTTP_ERROR;
```

WS_STATUS_HTTP_OK field

Error code indicating that the webservice HTTP request by the web services connector was successful.

Visual Basic syntax

```
Public Shared WS_STATUS_HTTP_OK As Integer
```

C# syntax

```
public static int WS_STATUS_HTTP_OK;
```

WS_STATUS_HTTP_RETRIES_EXCEEDED field

Error code indicating that the number of HTTP retries was exceeded the web services connector.

Visual Basic syntax

```
Public Shared WS_STATUS_HTTP_RETRIES_EXCEEDED As Integer
```

C# syntax

```
public static int WS_STATUS_HTTP_RETRIES_EXCEEDED;
```

WS_STATUS_SOAP_PARSE_ERROR field

Error code indicating that there was an error in the web services runtime or in the webservices connector in parsing a SOAP response or request.

Visual Basic syntax

```
Public Shared WS_STATUS_SOAP_PARSE_ERROR As Integer
```

C# syntax

```
public static int WS_STATUS_SOAP_PARSE_ERROR;
```

WSFaultException class

This class represents a SOAP Fault exception from the web service connector.

Visual Basic syntax

```
Public Class WSFaultException Inherits WSException
```

C# syntax

```
public class WSFaultException : WSException
```

Base classes

- "WSException class [QAnywhere .NET]" on page 314

Members

All members of WSFaultException class, including all inherited members.

Name	Description
WSException constructor	Constructs a new exception with the specified error message.

Name	Description
WSFaultException constructor	Constructs a new exception with the specified error message.
GetBaseException method (Inherited from System.Exception)	When overridden in a derived class, returns the System.Exception that is the root cause of one or more subsequent exceptions.
GetObjectData method (Inherited from System.Exception)	When overridden in a derived class, sets the System.Runtime.Serialization.SerializationInfo with information about the exception.
GetType method (Inherited from System.Exception)	Gets the runtime type of the current instance.
ToString method (Inherited from System.Exception)	Creates and returns a string representation of the current exception.
Data property (Inherited from System.Exception)	Gets a collection of key/value pairs that provide additional user-defined information about the exception.
ErrorCode property	The error code associated with this exception.
HelpLink property (Inherited from System.Exception)	Gets or sets a link to the help file associated with this exception.
HResult property (Inherited from System.Exception)	Gets or sets HRESULT, a coded numerical value that is assigned to a specific exception.
InnerException property (Inherited from System.Exception)	Gets the System.Exception instance that caused the current exception.
Message property (Inherited from System.Exception)	Gets a message that describes the current exception.
Source property (Inherited from System.Exception)	Gets or sets the name of the application or the object that causes the error.
StackTrace property (Inherited from System.Exception)	Gets a string representation of the frames on the call stack at the time the current exception was thrown.
TargetSite property (Inherited from System.Exception)	Gets the method that throws the current exception.
WS_STATUS_HTTP_ERROR field	Error code indicating that there was an error in the web service HTTP request made by the web services connector.

Name	Description
WS_STATUS_HTTP_OK field	Error code indicating that the webservice HTTP request by the web services connector was successful.
WS_STATUS_HTTP_RETRIES_EXCEEDED field	Error code indicating that the number of HTTP retries was exceeded the web services connector.
WS_STATUS_SOAP_PARSE_ERROR field	Error code indicating that there was an error in the web services runtime or in the webservices connector in parsing a SOAP response or request.

WSFaultException constructor

Constructs a new exception with the specified error message.

Visual Basic syntax

```
Public Sub New(ByVal msg As String)
```

C# syntax

```
public WSFaultException(string msg)
```

Parameters

- **msg** The error message.

WSListener interface

This class represents a listener for results of web service requests.

Visual Basic syntax

```
Public Interface WSListener
```

C# syntax

```
public interface WSListener
```

Members

All members of WSListener interface, including all inherited members.

Name	Description
OnException method	Called when an exception occurs during processing of the result of an asynchronous web service request.

Name	Description
OnResult method	Called with the result of an asynchronous web service request.

OnException method

Called when an exception occurs during processing of the result of an asynchronous web service request.

Visual Basic syntax

```
Public Sub OnException(
     ByVal e As WSException,
    ByVal wsResult As WSResult
)
```

C# syntax

```
public void OnException(WSException e, WSResult wsResult)
```

Parameters

- **e** The WSException that occurred during processing of the result.
- **wsResult** A WSResult, from which the request ID may be obtained. Values of this WSResult are not defined.

OnResult method

Called with the result of an asynchronous web service request.

Visual Basic syntax

```
Public Sub OnResult(ByVal wsResult As WSResult)
```

C# syntax

```
public void OnResult(WSResult wsResult)
```

Parameters

- **wsResult** The WSResult describing the result of a web service request.

WSResult class

This class represents the results of a web service request.

Visual Basic syntax

```
Public Class WSResult
```

C# syntax

```
public class WSResult
```

Members

All members of WSResult class, including all inherited members.

Name	Description
Acknowledge method	Acknowledges that this WSResult has been processed.
GetArrayValue method	Gets an array of complex types value from this WSResult.
GetBoolArrayValue method	Gets an array of bool values from this WSResult.
GetBooleanArrayValue method	Gets an array of Boolean values from this WSResult.
GetBooleanValue method	Gets a Boolean value from this WSResult.
GetBoolValue method	Gets a bool value from this WSResult.
GetByteArrayValue method	Gets an array of byte values from this WSResult.
GetByteValue method	Gets a byte value from this WSResult.
GetCharArrayValue method	Gets an array of char values from this WSResult.
GetCharValue method	Gets a char value from this WSResult.
GetDecimalArrayValue method	Gets an array of decimal values from this WSResult.
GetDecimalValue method	Gets a decimal value from this WSResult.
GetDoubleArrayValue method	Gets an array of double values from this WSResult.
GetDoubleValue method	Gets a double value from this WSResult.
GetErrorMessage method	Gets the error message.
GetFloatArrayValue method	Gets an array of float values from this WSResult.
GetFloatValue method	Gets a float value from this WSResult.
GetInt16ArrayValue method	Gets an array of Int16 values from this WSResult.
GetInt16Value method	Gets an Int16 value from this WSResult.
GetInt32ArrayValue method	Gets an array of Int32 values from this WSResult.
GetInt32Value method	Gets an Int32 value from this WSResult.

Name	Description
GetInt64ArrayValue method	Gets an array of Int64 values from this WSResult.
GetInt64Value method	Gets an Int64 value from this WSResult.
GetIntArrayValue method	Gets an array of int values from this WSResult.
GetIntValue method	Gets an int value from this WSResult.
GetLongArrayValue method	Gets an array of long values from this WSResult.
GetLongValue method	Gets a long value from this WSResult.
GetNullableBoolArrayValue method	Gets an array of bool values from this WSResult.
GetNullableBoolValue method	Gets a bool value from this WSResult.
GetNullableDecimalArrayValue method	Gets an array of NullableDecimal values from this WSResult.
GetNullableDecimalValue method	Gets a NullableDecimal value from this WSResult.
GetNullableDoubleArrayValue method	Gets an array of double values from this WSResult.
GetNullableDoubleValue method	Gets a double value from this WSResult.
GetNullableFloatArrayValue method	Gets an array of float values from this WSResult.
GetNullableFloatValue method	Gets a float value from this WSResult.
GetNullableIntArrayValue method	Gets an array of int values from this WSResult.
GetNullableIntValue method	Gets an int value from this WSResult.
GetNullableLongArrayValue method	Gets an array of long values from this WSResult.
GetNullableLongValue method	Gets an Int64 value from this WSResult.
GetNullableSByteArrayValue method	Gets an array of sbyte (signed byte) values from this WSResult.
GetNullableSByteValue method	Gets a sbyte (signed byte) value from this WSResult.
GetNullableShortArrayValue method	Gets an array of short values from this WSResult.
GetNullableShortValue method	Gets a short value from this WSResult.
GetObjectArrayValue method	Gets an array of Object values from this WSResult.

Name	Description
GetObjectValue method	Gets an object value from this WSResult.
GetRequestID method	Gets the request ID that this WSResult represents.
GetSByteArrayValue method	Gets an array of sbyte (signed byte) values from this WSResult.
GetSByteValue method	Gets an sbyte (signed byte) value from this WSResult.
GetShortArrayValue method	Gets an array of short values from this WSResult.
GetShortValue method	Gets a short value from this WSResult.
GetSingleArrayValue method	Gets an array of Single values from this WSResult.
GetSingleValue method	Gets a Single value from this WSResult.
GetStatus method	Gets the status of this WSResult.
GetStringArrayValue method	Gets an array of string values from this WSResult.
GetStringValue method	Gets a string value from this WSResult.
GetUIntArrayValue method	Gets an array of unsigned int values from this WSResult.
GetUIntValue method	Gets a unsigned int value from this WSResult.
GetULongArrayValue method	Gets an array of unsigned long values from this WSResult.
GetULongValue method	Gets a unsigned long value from this WSResult.
GetUShortArrayValue method	Gets an array of unsigned short values from this WSResult.
GetUShortValue method	Gets a unsigned short value from this WSResult.
GetValue method	Gets the value of a complex type from this WSResult.
SetLogger method	Turns debug on or off.

Remarks

A WSResult object is obtained in one of three ways:

- It is passed to the WSListener.onResult.
- It is returned by an asyncXYZ method of the service proxy generated by the compiler.
- It is obtained by calling WSBase.getResult with a specific request ID.

Acknowledge method

Acknowledges that this WSResult has been processed.

Visual Basic syntax

```
Public Sub Acknowledge()
```

C# syntax

```
public void Acknowledge()
```

Remarks

This method is only useful when an EXPLICIT_ACKNOWLEDGEMENT QAManager is being used.

GetArrayValue method

Gets an array of complex types value from this WSResult.

Visual Basic syntax

```
Public Function GetArrayValue(
     ByVal parentName As String
) As WSSerializable()
```

C# syntax

```
public WSSerializable[] GetArrayValue(string parentName)
```

Parameters

- **parentName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetBoolArrayValue method

Gets an array of bool values from this WSResult.

Visual Basic syntax

```
Public Function GetBoolArrayValue(
     ByVal elementName As String
) As Boolean()
```

C# syntax

```
public bool[] GetBoolArrayValue(string elementName)
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetBooleanArrayValue method

Gets an array of Boolean values from this WSResult.

Visual Basic syntax

```
Public Function GetBooleanArrayValue(
     ByVal elementName As String
) As System.Boolean()
```

C# syntax

```
public System.Boolean[] GetBooleanArrayValue(string elementName)
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetBooleanValue method

Gets a Boolean value from this WSResult.

Visual Basic syntax

```
Public Function GetBooleanValue(
     ByVal childName As String
) As System.Boolean
```

C# syntax

```
public System.Boolean GetBooleanValue(string childName)
```


Parameters

- **childName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetBoolValue method

Gets a bool value from this WSResult.

Visual Basic syntax

```
Public Function GetBoolValue(ByVal childName As String) As Boolean
```

C# syntax

```
public bool GetBoolValue(string childName)
```

Parameters

- **childName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetByteArrayValue method

Gets an array of byte values from this WSResult.

Visual Basic syntax

```
Public Function GetByteArrayValue(ByVal elementName As String) As Byte()
```

C# syntax

```
public byte[] GetByteArrayValue(string elementName)
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetByteValue method

Gets a byte value from this WSResult.

Visual Basic syntax

```
Public Function GetByteValue(ByVal childName As String) As Byte
```

C# syntax

```
public byte GetByteValue(string childName)
```

Parameters

- **childName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetCharArrayValue method

Gets an array of char values from this WSResult.

Visual Basic syntax

```
Public Function GetCharArrayValue(ByVal elementName As String) As Char()
```

C# syntax

```
public char[] GetCharArrayValue(string elementName)
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetCharValue method

Gets a char value from this WSResult.

Visual Basic syntax

```
Public Function GetCharValue(ByVal childName As String) As Char
```

C# syntax

```
public char GetCharValue(string childName)
```

Parameters

- **childName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetDecimalArrayValue method

Gets an array of decimal values from this WSResult.

Visual Basic syntax

```
Public Function GetDecimalArrayValue(
     ByVal elementName As String
) As Decimal()
```

C# syntax

```
public decimal[] GetDecimalArrayValue(string elementName)
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetDecimalValue method

Gets a decimal value from this WSResult.

Visual Basic syntax

```
Public Function GetDecimalValue(ByVal childName As String) As Decimal
```

C# syntax

```
public decimal GetDecimalValue(string childName)
```

Parameters

- **childName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetDoubleArrayValue method

Gets an array of double values from this WSResult.

Visual Basic syntax

```
Public Function GetDoubleArrayValue(
     ByVal elementName As String
) As Double()
```

C# syntax

```
public double[] GetDoubleArrayValue(string elementName)
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetDoubleValue method

Gets a double value from this WSResult.

Visual Basic syntax

```
Public Function GetDoubleValue(ByVal childName As String) As Double
```


C# syntax

```
public double GetDoubleValue(string childName)
```

Parameters

- **childName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetErrorMessage method

Gets the error message.

Visual Basic syntax

```
Public Function GetErrorMessage() As String
```

C# syntax

```
public string GetErrorMessage()
```

Returns

The error message.

GetFloatArrayValue method

Gets an array of float values from this WSResult.

Visual Basic syntax

```
Public Function GetFloatArrayValue(
     ByVal elementName As String
) As Single()
```

C# syntax

```
public float[] GetFloatArrayValue(string elementName)
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetFloatValue method

Gets a float value from this WSResult.

Visual Basic syntax

```
Public Function GetFloatValue(ByVal childName As String) As Single
```

C# syntax

```
public float GetFloatValue(string childName)
```

Parameters

- **childName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetInt16ArrayValue method

Gets an array of Int16 values from this WSResult.

Visual Basic syntax

```
Public Function GetInt16ArrayValue(
     ByVal elementName As String
) As System.Int16()
```

C# syntax

```
public System.Int16[] GetInt16ArrayValue(string elementName)
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetInt16Value method

Gets an Int16 value from this WSResult.

Visual Basic syntax

```
Public Function GetInt16Value(ByVal childName As String) As System.Int16
```

C# syntax

```
public System.Int16 GetInt16Value(string childName)
```

Parameters

- **childName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetInt32ArrayValue method

Gets an array of Int32 values from this WSResult.

Visual Basic syntax

```
Public Function GetInt32ArrayValue(
     ByVal elementName As String
) As System.Int32()
```

C# syntax

```
public System.Int32[] GetInt32ArrayValue(string elementName)
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetInt32Value method

Gets an Int32 value from this WSResult.

Visual Basic syntax

```
Public Function GetInt32Value(ByVal childName As String) As System.Int32
```

C# syntax

```
public System.Int32 GetInt32Value(string childName)
```

Parameters

- **childName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetInt64ArrayValue method

Gets an array of Int64 values from this WSResult.

Visual Basic syntax

```
Public Function GetInt64ArrayValue(
     ByVal elementName As String
) As System.Int64()
```

C# syntax

```
public System.Int64[] GetInt64ArrayValue(string elementName)
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetInt64Value method

Gets an Int64 value from this WSResult.

Visual Basic syntax

```
Public Function GetInt64Value(ByVal childName As String) As System.Int64
```


C# syntax

```
public System.Int64 GetInt64Value(string childName)
```

Parameters

- **childName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetIntArrayValue method

Gets an array of int values from this WSResult.

Visual Basic syntax

```
Public Function GetIntArrayValue(
     ByVal elementName As String
) As Integer()
```

C# syntax

```
public int[] GetIntArrayValue(string elementName)
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetIntValue method

Gets an int value from this WSResult.

Visual Basic syntax

```
Public Function GetIntValue(ByVal childName As String) As Integer
```

C# syntax

```
public int GetIntValue(string childName)
```

Parameters

- **childName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetLongArrayValue method

Gets an array of long values from this WSResult.

Visual Basic syntax

```
Public Function GetLongArrayValue(ByVal elementName As String) As Long()
```

C# syntax

```
public long[] GetLongArrayValue(string elementName)
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetLongValue method

Gets a long value from this WSResult.

Visual Basic syntax

```
Public Function GetLongValue(ByVal childName As String) As Long
```

C# syntax

```
public long GetLongValue(string childName)
```

Parameters

- **childName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetNullableBoolArrayValue method

Gets an array of bool values from this WSResult.

Visual Basic syntax

```
Public Function GetNullableBoolArrayValue(
     ByVal elementName As String
) As NullableBool()
```

C# syntax

```
public NullableBool[] GetNullableBoolArrayValue(string elementName)
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetNullableBoolValue method

Gets a bool value from this WSResult.

Visual Basic syntax

```
Public Function GetNullableBoolValue(
     ByVal childName As String
) As NullableBool
```

C# syntax

```
public NullableBool GetNullableBoolValue(string childName)
```

Parameters

- **childName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetNullableDecimalArrayValue method

Gets an array of NullableDecimal values from this WSResult.

Visual Basic syntax

```
Public Function GetNullableDecimalArrayValue(
     ByVal elementName As String
) As NullableDecimal()
```

C# syntax

```
public NullableDecimal[] GetNullableDecimalArrayValue(
     string elementName
)
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetNullableDecimalValue method

Gets a NullableDecimal value from this WSResult.

Visual Basic syntax

```
Public Function GetNullableDecimalValue(
     ByVal childName As String
) As NullableDecimal
```

C# syntax

```
public NullableDecimal GetNullableDecimalValue(string childName)
```

Parameters

- **childName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetNullableDoubleArrayValue method

Gets an array of double values from this WSResult.

Visual Basic syntax

```
Public Function GetNullableDoubleArrayValue(
     ByVal elementName As String
) As NullableDouble()
```

C# syntax

```
public NullableDouble[] GetNullableDoubleArrayValue(string elementName)
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetNullableDoubleValue method

Gets a double value from this WSResult.

Visual Basic syntax

```
Public Function GetNullableDoubleValue(
     ByVal childName As String
) As NullableDouble
```

C# syntax

```
public NullableDouble GetNullableDoubleValue(string childName)
```

Parameters

- **childName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetNullableFloatArrayValue method

Gets an array of float values from this WSResult.

Visual Basic syntax

```
Public Function GetNullableFloatArrayValue(
     ByVal elementName As String
) As NullableFloat()
```

C# syntax

```
public NullableFloat[] GetNullableFloatArrayValue(string elementName)
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetNullableFloatValue method

Gets a float value from this WSResult.

Visual Basic syntax

```
Public Function GetNullableFloatValue(
     ByVal childName As String
) As NullableFloat
```

C# syntax

```
public NullableFloat GetNullableFloatValue(string childName)
```

Parameters

- **childName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetNullableIntArrayValue method

Gets an array of int values from this WSResult.

Visual Basic syntax

```
Public Function GetNullableIntArrayValue(
     ByVal elementName As String
) As NullableInt()
```

C# syntax

```
public NullableInt[] GetNullableIntArrayValue(string elementName)
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetNullableIntValue method

Gets an int value from this WSResult.

Visual Basic syntax

```
Public Function GetNullableIntValue(
     ByVal childName As String
) As NullableInt
```

C# syntax

```
public NullableInt GetNullableIntValue(string childName)
```

Parameters

- **childName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetNullableLongArrayValue method

Gets an array of long values from this WSResult.

Visual Basic syntax

```
Public Function GetNullableLongArrayValue(
     ByVal elementName As String
) As NullableLong()
```

C# syntax

```
public NullableLong[] GetNullableLongArrayValue(string elementName)
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetNullableLongValue method

Gets an Int64 value from this WSResult.

Visual Basic syntax

```
Public Function GetNullableLongValue(
     ByVal childName As String
) As NullableLong
```

C# syntax

```
public NullableLong GetNullableLongValue(string childName)
```

Parameters

- **childName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetNullableSByteArrayValue method

Gets an array of sbyte (signed byte) values from this WSResult.

Visual Basic syntax

```
Public Function GetNullableSByteArrayValue(
     ByVal elementName As String
) As NullableSByte()
```

C# syntax

```
public NullableSByte[] GetNullableSByteArrayValue(string elementName)
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetNullableSByteValue method

Gets a sbyte (signed byte) value from this WSResult.

Visual Basic syntax

```
Public Function GetNullableSByteValue(
     ByVal childName As String
) As NullableSByte
```

C# syntax

```
public NullableSByte GetNullableSByteValue(string childName)
```

Parameters

- **childName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetNullableShortArrayValue method

Gets an array of short values from this WSResult.

Visual Basic syntax

```
Public Function GetNullableShortArrayValue(
     ByVal elementName As String
) As NullableShort()
```

C# syntax

```
public NullableShort[] GetNullableShortArrayValue(string elementName)
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetNullableShortValue method

Gets a short value from this WSResult.

Visual Basic syntax

```
Public Function GetNullableShortValue(
     ByVal childName As String
) As NullableShort
```

C# syntax

```
public NullableShort GetNullableShortValue(string childName)
```

Parameters

- **childName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetObjectArrayValue method

Gets an array of Object values from this WSResult.

Visual Basic syntax

```
Public Function GetObjectArrayValue(
     ByVal elementName As String
) As System.Object()
```

C# syntax

```
public System.Object[] GetObjectArrayValue(string elementName)
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetObjectValue method

Gets an object value from this WSResult.

Visual Basic syntax

```
Public Function GetObjectValue(
     ByVal childName As String
) As System.Object
```

C# syntax

```
public System.Object GetObjectValue(string childName)
```

Parameters

- **childName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetRequestID method

Gets the request ID that this WSResult represents.

Visual Basic syntax

```
Public Function GetRequestID() As String
```

C# syntax

```
public string GetRequestID()
```

Returns

The request ID.

Remarks

This request ID should be persisted between runs of the application if it is desired to obtain a WSResult corresponding to a web service request in a run of the application different from when the request was made.

GetSByteArrayValue method

Gets an array of sbyte (signed byte) values from this WSResult.

Visual Basic syntax

```
Public Function GetSByteArrayValue(
     ByVal elementName As String
) As SByte()
```

C# syntax

```
public sbyte[] GetSByteArrayValue(string elementName)
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetSByteValue method

Gets an sbyte (signed byte) value from this WSResult.

Visual Basic syntax

```
Public Function GetSByteValue(ByVal childName As String) As SByte
```

C# syntax

```
public sbyte GetSByteValue(string childName)
```

Parameters

- **childName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetShortArrayValue method

Gets an array of short values from this WSResult.

Visual Basic syntax

```
Public Function GetShortArrayValue(
     ByVal elementName As String
) As Short()
```

C# syntax

```
public short[] GetShortArrayValue(string elementName)
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetShortValue method

Gets a short value from this WSResult.

Visual Basic syntax

```
Public Function GetShortValue(ByVal childName As String) As Short
```

C# syntax

```
public short GetShortValue(string childName)
```

Parameters

- **childName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetSingleArrayValue method

Gets an array of Single values from this WSResult.

Visual Basic syntax

```
Public Function GetSingleArrayValue(
     ByVal elementName As String
) As System.Single()
```

C# syntax

```
public System.Single[] GetSingleArrayValue(string elementName)
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetSingleValue method

Gets a Single value from this WSResult.

Visual Basic syntax

```
Public Function GetSingleValue(
     ByVal childName As String
) As System.Single
```


C# syntax

```
public System.Single GetSingleValue(string childName)
```

Parameters

- **childName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetStatus method

Gets the status of this WSResult.

Visual Basic syntax

```
Public Function GetStatus() As WSStatus
```

C# syntax

```
public WSStatus GetStatus()
```

Returns

The status code.

See also

- "WSStatus enumeration [QAnywhere .NET]" on page 354

GetStringArrayValue method

Gets an array of string values from this WSResult.

Visual Basic syntax

```
Public Function GetStringArrayValue(
     ByVal elementName As String
) As String()
```

C# syntax

```
public string[] GetStringArrayValue(string elementName)
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetStringValue method

Gets a string value from this WSResult.

Visual Basic syntax

```
Public Function GetStringValue(ByVal childName As String) As String
```

C# syntax

```
public string GetStringValue(string childName)
```

Parameters

- **childName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetUIntArrayValue method

Gets an array of unsigned int values from this WSResult.

Visual Basic syntax

```
Public Function GetUIntArrayValue(
     ByVal elementName As String
) As UInteger()
```

C# syntax

```
public uint[] GetUIntArrayValue(string elementName)
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetUIntValue method

Gets a unsigned int value from this WSResult.

Visual Basic syntax

```
Public Function GetUIntValue(ByVal childName As String) As UInteger
```

C# syntax

```
public uint GetUIntValue(string childName)
```

Parameters

- **childName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetULongArrayValue method

Gets an array of unsigned long values from this WSResult.

Visual Basic syntax

```
Public Function GetULongArrayValue(
     ByVal elementName As String
) As ULong()
```

C# syntax

```
public ulong[] GetULongArrayValue(string elementName)
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetULongValue method

Gets a unsigned long value from this WSResult.

Visual Basic syntax

```
Public Function GetULongValue(ByVal childName As String) As ULong
```

C# syntax

```
public ulong GetULongValue(string childName)
```

Parameters

- **childName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetUShortArrayValue method

Gets an array of unsigned short values from this WSResult.

Visual Basic syntax

```
Public Function GetUShortArrayValue(
     ByVal elementName As String
) As UShort()
```

C# syntax

```
public ushort[] GetUShortArrayValue(string elementName)
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetUShortValue method

Gets a unsigned short value from this WSResult.

Visual Basic syntax

```
Public Function GetUShortValue(ByVal childName As String) As UShort
```

C# syntax

```
public ushort GetUShortValue(string childName)
```

Parameters

- **childName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

GetValue method

Gets the value of a complex type from this WSResult.

Visual Basic syntax

```
Public Function GetValue(ByVal childName As String) As Object
```

C# syntax

```
public object GetValue(string childName)
```

Parameters

- **childName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **iAnywhere.QAnywhere.WS.WSException** Thrown if there is a problem getting the value.

SetLogger method

Turns debug on or off.

Visual Basic syntax

```
Public Sub SetLogger(ByVal wsLogger As WSLogger)
```

C# syntax

```
public void SetLogger(WSLogger wsLogger)
```

WSStatus enumeration

This class defines codes for the status of a web service request.

Visual Basic syntax

```
Public Enum WSStatus
```

C# syntax

```
public enum WSStatus
```

Members

Member name	Description
STATUS_SUCCESS	The request was successful.
STATUS_ERROR	There was an error processing the request.
STATUS_QUEUED	The request has been queued for delivery to the server. The final status of the request is not known yet.
STATUS_RESULT_AVAILABLE	The result of the request is available.

QAnywhere C++ API reference for clients

The QAnywhere C++ API does not support UltraLite databases.

Header file

```
qa.hpp
```

AcknowledgementMode class

Indicates how messages should be acknowledged by QAnywhere client applications.

Syntax

```
public class AcknowledgementMode
```

Members

All members of AcknowledgementMode class, including all inherited members.

Name	Description
EXPLICIT_ACKNOWLEDGEMENT variable	Indicates that received messages are acknowledged using one of the QAManager acknowledge methods.
IMPLICIT_ACKNOWLEDGEMENT variable	Indicates that all messages are acknowledged as soon as they are received by a client application.
TRANSACTIONAL variable	Indicates that messages are only acknowledged as part of the ongoing transaction.

Remarks

The IMPLICIT_ACKNOWLEDGEMENT and EXPLICIT_ACKNOWLEDGEMENT modes are assigned to a QAManager instance using the QAManager::open() method. The TRANSACTIONAL mode is implicitly assigned to QATransactionalManager instances.

For more information, see "Initializing a QAnywhere API" on page 52.

In implicit acknowledgement mode, messages are acknowledged as soon as they are received by a client application. In explicit acknowledgement mode, you must call one of the QAManager acknowledgement methods. In transactional mode, you must call the QATransactionalManager::commit() method to acknowledge all outstanding messages. The server propagates all status changes from client to client.

For more information, see "Receiving messages synchronously" on page 68 and "Receiving messages asynchronously" on page 69.

For transactional messaging, use the QATransactionalManager. In this case, you use the QATransactionalManager::commit method to acknowledge messages belonging to a transaction.

You can determine the mode of a QAManagerBase instance using the QAManagerBase::Mode property.

See also

- "QAManager class [QAnywhere C++]" on page 388
- "QATransactionalManager class [QAnywhere C++]" on page 457
- "QAManagerBase class [QAnywhere C++]" on page 394

EXPLICIT_ACKNOWLEDGEMENT variable

Indicates that received messages are acknowledged using one of the QAManager acknowledge methods.

Syntax

```
public static const qa_short EXPLICIT_ACKNOWLEDGEMENT;
```

IMPLICIT_ACKNOWLEDGEMENT variable

Indicates that all messages are acknowledged as soon as they are received by a client application.

Syntax

```
public static const qa_short IMPLICIT_ACKNOWLEDGEMENT;
```

Remarks

If you receive messages synchronously, messages are acknowledged as soon as the QAManagerBase::getMessage method returns. If you receive messages asynchronously, the message is acknowledged as soon as the event handling function returns.

TRANSACTIONAL variable

Indicates that messages are only acknowledged as part of the ongoing transaction.

Syntax

```
public static const qa_short TRANSACTIONAL;
```

Remarks

This mode is automatically assigned to QATransactionalManager instances.

MessageProperties class

Provides fields storing standard message property names.

Syntax

```
public class MessageProperties
```

Members

All members of MessageProperties class, including all inherited members.

Name	Description
ADAPTER variable	This property name refers to the currently active network adapter that is being used to connect to the QAnywhere server.
ADAPTERS variable	This property name refers to a delimited list of network adapters that can be used to connect to the QAnywhere server.
DELIVERY_COUNT variable	This property name refers to the number of attempts that have been made so far to deliver the message.

Name	Description
IP variable	This property name refers to the IP address of the currently active network adapter that is being used to connect to the QAnywhere server.
MAC variable	This property name refers to the MAC address of the currently active network adapter that is being used to connect to the QAnywhere server.
MSG_TYPE variable	This property name refers to MessageType enumeration values associated with a QAnywhere message.
NETWORK_STATUS variable	This property name refers to the state of the network connection.
ORIGINATOR variable	This property name refers to the message store ID of the originator of the message.
RAS variable	This property name refers to the currently active RAS name that is being used to connect to the QAnywhere server.
RASNAMES variable	This property name refers to a delimited list of RAS entry names that can be used to connect to the QAnywhere server.
STATUS variable	This property name refers to the current status of the message.
STATUS_TIME variable	This property name refers to the time at which the message received its current status.
TRANSMISSION_STATUS variable	This property name refers to the current transmission status of the message.

Remarks

The MessageProperties class provides standard message property names. You can pass MessageProperties fields to QAMessage methods used to get and set message properties.

For more information, see "QAnywhere messages" on page 13.

See also

- "QAMessage class [QAnywhere C++]" on page 429

Example

For example, assume you have the following QATextMessage instance:

```
QATextMessage * t_msg;
```

The following example gets the value corresponding to MessageProperties::MSG_TYPE using the QAMessage::getIntProperty method. The MessageType enumeration maps the integer result to an appropriate message type.

```
int msg_type;
t_msg->getIntProperty( MessageProperties::MSG_TYPE, &msg_type)
```

The following example, evaluates the message type and RAS names using MessageProperties::MSG_TYPE and MessageProperties::RASNAMES respectively.

```
void SystemQueueListener::onMessage(QAMessage * msg) {
       QATextMessage *        t_msg;
       TCHAR                buffer[512];
       int                    len;
       int                    msg_type;

       t_msg = msg->castToTextMessage();
       if( t_msg != NULL ) {
           t_msg->getIntProperty( MessageProperties::MSG_TYPE, &msg_type );

        if( msg_type == MessageType::NETWORK_STATUS_NOTIFICATION ) {
            // get RAS names using MessageProperties::RASNAMES
            len = t_msg-
>getStringProperty(MessageProperties::RASNAMES,buffer,sizeof(buffer));
        }
        //...
    }
}
```

ADAPTER variable

This property name refers to the currently active network adapter that is being used to connect to the QAnywhere server.

Syntax

```
public static const qa_string ADAPTER;
```

Remarks

It is used for system queue messages.

The value of this field is "ias_Network.Adapter".

Pass MessageProperties::ADAPTER as the first parameter to the QAMessage::getStringProperty method to access the associated message property.

For more information, see “Message properties” on page 658.

See also

- “QAMessage.getStringProperty method [QAnywhere C++]” on page 441

ADAPTERS variable

This property name refers to a delimited list of network adapters that can be used to connect to the QAnywhere server.

Syntax

```
public static const qa_string ADAPTERS;
```

Remarks

It is used for system queue messages.

The value of this field is "ias_Adapters".

Pass MessageProperties::ADAPTERS as the first parameter to the QAMessage::getStringProperty method to access the associated message property.

For more information, see "Message properties" on page 658.

See also

- "QAMessage.getStringProperty method [QAnywhere C++]" on page 441

DELIVERY_COUNT variable

This property name refers to the number of attempts that have been made so far to deliver the message.

Syntax

```
public static const qa_string DELIVERY_COUNT;
```

Remarks

The value of this field is "ias_DeliveryCount".

Pass MessageProperties::DELIVERY_COUNT as the first parameter in the QAMessage::setStringProperty method or the QAMessage::getStringProperty method to access the associated message property.

See also

- "QAMessage.setStringProperty method [QAnywhere C++]" on page 450
- "QAMessage.getStringProperty method [QAnywhere C++]" on page 441

IP variable

This property name refers to the IP address of the currently active network adapter that is being used to connect to the QAnywhere server.

Syntax

```
public static const qa_string IP;
```

Remarks

It is used for system queue messages.

The value of this field is "ias_Network.IP".

Pass MessageProperties::IP as the first parameter to the QAMessage::getStringProperty method to access the associated message property.

For more information, see "Message properties" on page 658.

See also

- "QAMessage.getStringProperty method [QAnywhere C++]" on page 441

MAC variable

This property name refers to the MAC address of the currently active network adapter that is being used to connect to the QAnywhere server.

Syntax

```
public static const qa_string MAC;
```

Remarks

It is used for system queue messages.

The value of this field is "ias_Network.MAC".

Pass MessageProperties::MAC as the first parameter to the QAMessage::getStringProperty method to access the associated message property.

For more information, see "Message properties" on page 658.

See also

- "QAMessage.getStringProperty method [QAnywhere C++]" on page 441

MSG_TYPE variable

This property name refers to MessageType enumeration values associated with a QAnywhere message.

Syntax

```
public static const qa_string MSG_TYPE;
```

Remarks

The value of this field is "ias_MessageType". Pass MessageProperties::MSG_TYPE as the first parameter in the QAMessage::setIntProperty method or the QAMessage::getIntProperty method to determine the associated property.

See also

- "MessageType class [QAnywhere C++]" on page 365
- "QAMessage.setIntProperty method [QAnywhere C++]" on page 447
- "QAMessage.getIntProperty method [QAnywhere C++]" on page 437

NETWORK_STATUS variable

This property name refers to the state of the network connection.

Syntax

```
public static const qa_string NETWORK_STATUS;
```

Remarks

The value of this field is "ias_NetworkStatus".

The value of this property is 1 if the network is accessible and 0 otherwise. The network status is used for system queue messages (for example, network status changes).

For more information, see "Message properties" on page 658.

Pass MessageProperties::NETWORK_STATUS as the first parameter in the QAMessage::setStringProperty method or the QAMessage::getStringProperty method to access the associated message property.

See also

- "QAMessage.setStringProperty method [QAnywhere C++]" on page 450
- "QAMessage.getStringProperty method [QAnywhere C++]" on page 441

ORIGINATOR variable

This property name refers to the message store ID of the originator of the message.

Syntax

```
public static const qa_string ORIGINATOR;
```

Remarks

The value of this field is "ias_Originator".

Pass MessageProperties::ORIGINATOR as the first parameter in the QAMessage::setStringProperty method or the QAMessage::getStringProperty method to access the associated message property.

See also

- "QAMessage.setStringProperty method [QAnywhere C++]" on page 450
- "QAMessage.getStringProperty method [QAnywhere C++]" on page 441

RAS variable

This property name refers to the currently active RAS name that is being used to connect to the QAnywhere server.

Syntax

```
public static const qa_string RAS;
```

Remarks

It is used for system queue messages.

The value of this field is "ias_Network.RAS".

Pass MessageProperties::RAS as the first parameter to the QAMessage::getStringProperty method to access the associated message property.

For more information, see "Message properties" on page 658.

See also

- "QAMessage.getStringProperty method [QAnywhere C++]" on page 441

RASNAMES variable

This property name refers to a delimited list of RAS entry names that can be used to connect to the QAnywhere server.

Syntax

```
public static const qa_string RASNAMES;
```

Remarks

It is used for system queue messages.

The value of this field is "ias_RASNames".

For more information, see "Message properties" on page 658.

Pass MessageProperties::RASNAMES as the first parameter in the QAMessage::setStringProperty method or the QAMessage::getStringProperty method to access the associated message property.

See also

- "QAMessage.setStringProperty method [QAnywhere C++]" on page 450
- "QAMessage.getStringProperty method [QAnywhere C++]" on page 441
- "QAMessage.setIntProperty method [QAnywhere C++]" on page 447
- "QAMessage.getIntProperty method [QAnywhere C++]" on page 437

STATUS variable

This property name refers to the current status of the message.

Syntax

```
public static const qa_string STATUS;
```

Remarks

For a list of values, see the StatusCodes. The value of this field is "ias_Status".

Pass MessageProperties::STATUS as the first parameter in the QAMessage::setIntProperty method or the QAMessage::getIntProperty method to access the associated message property.

See also

- "StatusCodes class [QAnywhere C++]" on page 464
- "QAMessage.setIntProperty method [QAnywhere C++]" on page 447
- "QAMessage.getIntProperty method [QAnywhere C++]" on page 437

STATUS_TIME variable

This property name refers to the time at which the message received its current status.

Syntax

```
public static const qa_string STATUS_TIME;
```

Remarks

It is in units that are natural for the platform. For Windows/PocketPC platforms, the timestamp is the SYSTEMTIME, converted to a FILETIME, which is copied to a qa_long value. It is a local time. The value of this field is "ias_StatusTime".

Pass MessageProperties::STATUS_TIME as the first parameter in the QAMessage::getLongProperty method to access the associated read-only message property.

See also

- "QAMessage.getLongProperty method [QAnywhere C++]" on page 437

TRANSMISSION_STATUS variable

This property name refers to the current transmission status of the message.

Syntax

```
public static const qa_string TRANSMISSION_STATUS;
```

Remarks

For a list of values, see the StatusCodes.

The value of this field is "ias_TransmissionStatus".

Pass MessageProperties::TRANSMISSION_STATUS as the first parameter in the QAMessage::setIntProperty method or the QAMessage::getIntProperty method to access the associated message property.

See also

- "StatusCodes class [QAnywhere C++]" on page 464
- "QAMessage.setIntProperty method [QAnywhere C++]" on page 447
- "QAMessage.getIntProperty method [QAnywhere C++]" on page 437

MessageStoreProperties class

The MessageStoreProperties class provides standard message property names.

Syntax

```
public class MessageStoreProperties
```

Members

All members of MessageStoreProperties class, including all inherited members.

Name	Description
MAX_DELIVERY_ATTEMPTS variable	This property name refers to the maximum number of times that a message can be received, without explicit acknowledgement, before its status is set to StatusCodes::UNRECEIVABLE.

Remarks

You can pass MessageStoreProperties fields to QAManagerBase methods used to get and set predefined or custom message store properties.

For more information, see "Client message store properties" on page 26.

MAX_DELIVERY_ATTEMPTS variable

This property name refers to the maximum number of times that a message can be received, without explicit acknowledgement, before its status is set to StatusCodes::UNRECEIVABLE.

Syntax

```
public static const qa_string MAX_DELIVERY_ATTEMPTS;
```


Remarks

The value of this field is "ias_MaxDeliveryAttempts".

See also

- "StatusCodes class [QAnywhere C++]" on page 464

MessageType class

Defines constant values for the MessageProperties::MSG_TYPE message property.

Syntax

```
public class MessageType
```

Members

All members of MessageType class, including all inherited members.

Name	Description
NETWORK_STATUS_NOTIFICATION variable	Identifies a QAnywhere system message used to notify QAnywhere client applications of network status changes.
PUSH_NOTIFICATION variable	Identifies a QAnywhere system message used to notify QAnywhere client applications of push notifications.
REGULAR variable	If no message type property exists then the message type is assumed to be REGULAR.

Remarks

The following example shows the onSystemMessage method which is used to handle QAnywhere system messages.

The message type is compared to MessageType.NETWORK_STATUS_NOTIFICATION.

```
  void SystemQueueListener::onMessage(QAMessage * msg) {
      QATextMessage *        t_msg;
      TCHAR                buffer[512];
      int                    len;
      int                    msg_type;

      t_msg = msg->castToTextMessage();
      if( t_msg != NULL ) {
          t_msg->getIntProperty( MessageProperties::MSG_TYPE, &msg_type );

             if( msg_type == MessageType::NETWORK_STATUS_NOTIFICATION ) {
                 // get network names using MessageProperties::NETWORK
              len = t_msg-
>getStringProperty(MessageProperties::NETWORK,buffer,sizeof(buffer));
          }
          //...
```

```
        }
    }
```

NETWORK_STATUS_NOTIFICATION variable

Identifies a QAnywhere system message used to notify QAnywhere client applications of network status changes.

Syntax

```
public static const qa_int NETWORK_STATUS_NOTIFICATION;
```

Remarks

Network status changes apply to the device receiving the system message. Use the MessageProperties::ADAPTER, MessageProperties::NETWORK, and MessageProperties::NETWORK_STATUS fields to identify new network status information.

For more information, see "Predefined message properties" on page 659.

PUSH_NOTIFICATION variable

Identifies a QAnywhere system message used to notify QAnywhere client applications of push notifications.

Syntax

```
public static const qa_int PUSH_NOTIFICATION;
```

Remarks

If you use the on-demand QAnywhere Agent policy, a typical response is to call the QAManagerBase::triggerSendReceive() method to receive messages waiting with the central message server.

For more information, see "Predefined message properties" on page 659.

REGULAR variable

If no message type property exists then the message type is assumed to be REGULAR.

Syntax

```
public static const qa_int REGULAR;
```

Remarks

This type of message is not treated specially by the message system.

QABinaryMessage class

A QABinaryMessage object is used to send a message containing a stream of uninterpreted bytes.

Syntax

```
public class QABinaryMessage : QAMessage
```

Base classes

- "QAMessage class [QAnywhere C++]" on page 429

Members

All members of QABinaryMessage class, including all inherited members.

Name	Description
QABinaryMessage deconstructor	Virtual destructor.
beginEnumPropertyNames method	Begins an enumeration of message property names.
castToBinaryMessage method	Casts this QAMessage to a QABinaryMessage.
castToTextMessage method	Casts this QAMessage to a QATextMessage.
clearProperties method	Clears a message's properties.
endEnumPropertyNames method	Frees the resources associated with a message property name enumeration.
getAddress method	Gets the destination address for the QAMessage instance.
getBodyLength method	Returns the size of the message body in bytes.
getBooleanProperty method	Gets the value of the qa_bool property with the specified name.
getByteProperty method	Gets the value of the qa_byte property with the specified name.
getDoubleProperty method	Gets the value of the qa_double property with the specified name.
getExpiration method	Gets the message's expiration time.
getFloatProperty method	Gets the value of the qa_float property with the specified name.
getInReplyToID method	Gets the ID of the message that this message is in reply to.

Name	Description
getIntProperty method	Gets the value of the qa_int property with the specified name.
getLongProperty method	Gets the value of the qa_long property with the specified name.
getMessageID method	Gets the message ID.
getPriority method	Gets the message priority level.
getPropertyType method	Returns the type of a property with the given name.
getRedelivered method	Indicates whether the message has been previously received but not acknowledged.
getReplyToAddress method	Gets the address to which a reply to this message should be sent.
getShortProperty method	Gets the value of the qa_short property with the specified name.
getStringProperty method	Gets the value of the qa_string property with the specified name.
getTimestamp method	Gets the message timestamp.
getTimestampAsString method	Gets the message timestamp as a formatted string.
nextPropertyName method	Returns the message property name for the given enumeration, returning -1 if there are no more property names.
propertyExists method	Indicates whether a property value exists.
readBinary method	Reads a specified number of bytes starting from the unread portion of the QABinaryMessage instance's message body.
readBoolean method	Reads a boolean value starting from the unread portion of the QABinaryMessage instance's message body.
readByte method	Reads a signed 8-bit value starting from the unread portion of the QABinaryMessage instance's message body.
readChar method	Reads a character value starting from the unread portion of the QABinaryMessage instance's message body.
readDouble method	Reads a double value starting from the unread portion of the QABinaryMessage instance's message body.

Name	Description
readFloat method	Reads a float value starting from the unread portion of the QABinaryMessage instance's message body.
readInt method	Reads a signed 32-bit integer value starting from the unread portion of the QABinaryMessage instance's message body.
readLong method	Reads a signed 64-bit integer value starting from the unread portion of the QABinaryMessage instance's message body.
readShort method	Reads a signed 16-bit value starting from the unread portion of the QABinaryMessage instance's message body.
readString method	Reads a string value starting from the unread portion of the QABinaryMessage instance's message body.
reset method	Resets a message so that the reading of values starts from the beginning of the message body.
setAddress method	Sets the destination address for this message.
setBooleanProperty method	Sets the qa_bool property with the specified name to the specified value.
setByteProperty method	Sets a qa_byte property with the specified name to the specified value.
setDoubleProperty method	Sets the qa_double property with the specified name to the specified value.
setFloatProperty method	Sets the qa_float property with the specified name to the specified value.
setInReplyToID method	Sets the In-Reply-To ID for the message.
setIntProperty method	Sets the qa_int property with the specified name to the specified value.
setLongProperty method	Sets the qa_long property with the specified name to the specified value.
setMessageID method	Sets the message ID.
setPriority method	Sets the priority level for this message.
setRedelivered method	Sets an indication of whether this message was redelivered.

Name	Description
setReplyToAddress method	Sets the address to which a reply to this message should be sent.
setShortProperty method	Sets e qa_short property with the specified name to the specified value.
setStringProperty method	Sets a qa_string property with the specified name to the specified value.
setTimestamp method	Sets the message timestamp.
writeBinary method	Appends a byte array value to the QABinaryMessage instance's message body.
writeBoolean method	Appends a boolean value to the QABinaryMessage instance's message body.
writeByte method	Appends a byte value to the QABinaryMessage instance's message body.
writeChar method	Appends a char value to the QABinaryMessage instance's message body.
writeDouble method	Appends a double value to the QABinaryMessage instance's message body.
writeFloat method	Appends a float value to the QABinaryMessage instance's message body.
writeInt method	Appends an integer value to the QABinaryMessage instance's message body.
writeLong method	Appends a long value to the QABinaryMessage instance's message body.
writeShort method	Appends a short value to the QABinaryMessage instance's message body.
writeString method	Appends a string value to the QABinaryMessage instance's message body.
DEFAULT_PRIORITY variable	The default message priority.
DEFAULT_TIME_TO_LIVE variable	The default message time-to-live value.

Remarks

It inherits from the QAMessage class and adds a bytes message body. QABinaryMessage provides a variety of methods to read from and write to the bytes message body.

When the message is first created, the body of the message is write-only. After a message has been sent, the client that sent it can retain and modify it without affecting the message that has been sent. The same message object can be sent multiple times.

When a message is received, the provider has called the QABinaryMessage::reset method so that the message body is in read-only mode and reading of values starts from the beginning of the message body. If a client attempts to write a message in read-only mode, a COMMON_MSG_NOT_WRITEABLE_ERROR is set.

Example

The following example uses the QABinaryMessage::writeString method to write the string "Q" followed by the string "Anywhere" to a QABinaryMessage instance's message body.

```
// Create a binary message instance.
QABinaryMessage * binary_message;
binary_message = qa_manager->createBinaryMessage();

// Set optional message properties.
binary_message->setReplyToAddress( "my-queue-name" );

// Write to the message body.
binary_message->writeString("Q");
binary_message->writeString("Anywhere");

// Put the message in the local database, ready for sending.

if( !qa_manager->putMessage( "store-id\\queue-name", msg ) ) {
    handleError();
}
```

On the receiving end, the first QABinaryMessage::readString() invocation returns "Q" and the next QABinaryMessage::readString() invocation returns "Anywhere".

The message is sent by the QAnywhere Agent.

For more information, see "Determining when message transmission should occur on the client" on page 46 and "How to write QAnywhere client applications" on page 49.

QABinaryMessage deconstructor

Virtual destructor.

Syntax

```
public virtual ~QABinaryMessage()
```

getBodyLength method

Returns the size of the message body in bytes.

Syntax

```
public virtual qa_long getBodyLength()
```

Returns

The size of the message body in bytes.

readBinary method

Reads a specified number of bytes starting from the unread portion of the QABinaryMessage instance's message body.

Syntax

```
public virtual qa_int readBinary(qa_bytes value, qa_int length)
```

Parameters

- **value** The buffer into which the data is read.
- **length** The maximum number of bytes to read.

Returns

The total number of bytes read into the buffer, or -1 if there is no more data because the end of the stream has been reached.

See also

- "QABinaryMessage.writeBinary method [QAnywhere C++]" on page 376

readBoolean method

Reads a boolean value starting from the unread portion of the QABinaryMessage instance's message body.

Syntax

```
public virtual qa_bool readBoolean(qa_bool * value)
```

Parameters

- **value** The destination of the qa_bool value read from the bytes message stream.

Returns

True if and only if the operation succeeded.

See also

- "QABinaryMessage.writeBoolean method [QAnywhere C++]" on page 377

readByte method

Reads a signed 8-bit value starting from the unread portion of the QABinaryMessage instance's message body.

Syntax

```
public virtual qa_bool readByte(qa_byte * value)
```

Parameters

- **value** The destination of the qa_byte value read from the bytes message stream.

Returns

True if and only if the operation succeeded.

See also

- "QABinaryMessage.writeByte method [QAnywhere C++]" on page 377

readChar method

Reads a character value starting from the unread portion of the QABinaryMessage instance's message body.

Syntax

```
public virtual qa_bool readChar(qa_char * value)
```

Parameters

- **value** The destination of the qa_char value read from the bytes message stream.

Returns

The character value read.

See also

- "QABinaryMessage.writeChar method [QAnywhere C++]" on page 377

readDouble method

Reads a double value starting from the unread portion of the QABinaryMessage instance's message body.

Syntax

```
public virtual qa_bool readDouble(qa_double * value)
```

Parameters

- **value** The destination of the double value read from the bytes message stream.

Returns

True if and only if the operation succeeded.

See also

- "QABinaryMessage.writeDouble method [QAnywhere C++]" on page 378

readFloat method

Reads a float value starting from the unread portion of the QABinaryMessage instance's message body.

Syntax

```
public virtual qa_bool readFloat(qa_float * value)
```

Parameters

- **value** The destination of the float value read from the bytes message stream.

Returns

True if and only if the operation succeeded.

See also

- "QABinaryMessage.writeFloat method [QAnywhere C++]" on page 378

readInt method

Reads a signed 32-bit integer value starting from the unread portion of the QABinaryMessage instance's message body.

Syntax

```
public virtual qa_bool readInt(qa_int * value)
```

Parameters

- **value** The destination of the qa_int value read from the bytes message stream.

Returns

True if and only if the operation succeeded.

See also

- "QABinaryMessage.writeInt method [QAnywhere C++]" on page 379

readLong method

Reads a signed 64-bit integer value starting from the unread portion of the QABinaryMessage instance's message body.

Syntax

```
public virtual qa_bool readLong(qa_long * value)
```

Parameters

- **value** The destination of the long value read from the bytes message stream.

Returns

True if and only if the operation succeeded.

See also

- "QABinaryMessage.writeLong method [QAnywhere C++]" on page 379

readShort method

Reads a signed 16-bit value starting from the unread portion of the QABinaryMessage instance's message body.

Syntax

```
public virtual qa_bool readShort(qa_short * value)
```

Parameters

- **value** The destination of the qa_short value read from the bytes message stream.

Returns

True if and only if the operation succeeded.

See also

- "QABinaryMessage.writeShort method [QAnywhere C++]" on page 379

readString method

Reads a string value starting from the unread portion of the QABinaryMessage instance's message body.

Syntax

```
public virtual qa_int readString(qa_string dest, qa_int maxLen)
```

Parameters

- **dest** The destination of the qa_string value read from the bytes message stream.
- **maxLen** The maximum number of characters to read, including the null terminator character.

Returns

The total number of non-null qa_chars read into the buffer, -1 if there is no more data or an error occurred, or -2 if the buffer is too small.

See also

- "QABinaryMessage.writeString method [QAnywhere C++]" on page 380

reset method

Resets a message so that the reading of values starts from the beginning of the message body.

Syntax

```
public virtual void reset()
```

Remarks

The reset method also puts the QABinaryMessage message body in read-only mode.

writeBinary method

Appends a byte array value to the QABinaryMessage instance's message body.

Syntax

```
public virtual void writeBinary(
     qa_const_bytes value,
    qa_int offset,
    qa_int length
)
```

Parameters

- **value** The byte array value to write to the message body.
- **offset** The offset within the byte array to begin writing.
- **length** The number of bytes to write.

See also

- "QABinaryMessage.readBinary method [QAnywhere C++]" on page 372

writeBoolean method

Appends a boolean value to the QABinaryMessage instance's message body.

Syntax

```
public virtual void writeBoolean(qa_bool value)
```

Parameters

- **value** The boolean value to write to the message body.

Remarks

The boolean is represented as a one-byte value. True is represented as 1; false is represented as 0.

See also

- "QABinaryMessage.readBoolean method [QAnywhere C++]" on page 372

writeByte method

Appends a byte value to the QABinaryMessage instance's message body.

Syntax

```
public virtual void writeByte(qa_byte value)
```

Parameters

- **value** The byte array value to write to the message body.

Remarks

The byte is represented as a one-byte value.

See also

- "QABinaryMessage.readByte method [QAnywhere C++]" on page 373

writeChar method

Appends a char value to the QABinaryMessage instance's message body.

Syntax

```
public virtual void writeChar(qa_char value)
```

Parameters

- **value** the char value to write to the message body.

Remarks

The char parameter is represented as a two-byte value and the high order byte is appended first.

See also

- "QABinaryMessage.readChar method [QAnywhere C++]" on page 373

writeDouble method

Appends a double value to the QABinaryMessage instance's message body.

Syntax

```
public virtual void writeDouble(qa_double value)
```

Parameters

- **value** The double value to write to the message body.

Remarks

The double parameter is converted to a representative eight-byte long value. Higher order bytes are appended first.

See also

- "QABinaryMessage.readDouble method [QAnywhere C++]" on page 373

writeFloat method

Appends a float value to the QABinaryMessage instance's message body.

Syntax

```
public virtual void writeFloat(qa_float value)
```

Parameters

- **value** The float value to write to the message body.

Remarks

The float parameter is converted to a representative 4-byte integer and the higher order bytes are appended first.

See also

- "QABinaryMessage.readFloat method [QAnywhere C++]" on page 374

writeInt method

Appends an integer value to the QABinaryMessage instance's message body.

Syntax

```
public virtual void writeInt(qa_int value)
```

Parameters

- **value** the int value to write to the message body.

Remarks

The integer parameter is represented as a four-byte value and higher order bytes are appended first.

See also

- "QABinaryMessage.readInt method [QAnywhere C++]" on page 374

writeLong method

Appends a long value to the QABinaryMessage instance's message body.

Syntax

```
public virtual void writeLong(qa_long value)
```

Parameters

- **value** The long value to write to the message body.

Remarks

The long parameter is represented as an eight-byte value and higher order bytes are appended first.

See also

- "QABinaryMessage.readLong method [QAnywhere C++]" on page 375

writeShort method

Appends a short value to the QABinaryMessage instance's message body.

Syntax

```
public virtual void writeShort(qa_short value)
```

Parameters

- **value** The short value to write to the message body.

Remarks

The short parameter is represented as a two-byte value and the higher order byte is appended first.

See also

- "QABinaryMessage.readShort method [QAnywhere C++]" on page 375

writeString method

Appends a string value to the QABinaryMessage instance's message body.

Syntax

```
public virtual void writeString(qa_const_string value)
```

Parameters

- **value** The string value to write to the message body.

Remarks

Note
The receiving application needs to invoke QABinaryMessage::readString for each writeString invocation.

Note
The UTF-8 representation of the string can be at most 32767 bytes.

See also

- "QABinaryMessage.readString method [QAnywhere C++]" on page 375

QAError class

This class defines error constants associated with a QAnywhere client application.

Syntax

```
public class QAError
```

Members

All members of QAError class, including all inherited members.

Name	Description
COMMON_ALREADY_OPEN_ERROR variable	The QAManager is already open.
COMMON_GET_INIT_FILE_ERROR variable	Unable to access the client properties file.

Name	Description
COMMON_GET_PROPERTY_ERROR variable	Error retrieving the property.
COMMON_GETQUEUEDEPTH_ERROR variable	Error getting queue depth.
COMMON_GETQUEUEDEPTH_ERROR_INVALID_ARG variable	Cannot use QAManagerBase.getQueueDepth on a given destination when filter is ALL.
COMMON_GETQUEUEDEPTH_ERROR_NO_STORE_ID variable	Cannot use QAManagerBase.getQueueDepth when the message store ID has not been set.
COMMON_INIT_ERROR variable	Initialization error.
COMMON_INIT_THREAD_ERROR variable	Error initializing the background thread.
COMMON_INVALID_PROPERTY variable	There is an invalid property in the client properties file.
COMMON_MSG_ACKNOWLEDGE_ERROR variable	Error acknowledging the message.
COMMON_MSG_CANCEL_ERROR variable	Error canceling message.
COMMON_MSG_CANCEL_ERROR_SENT variable	Error canceling message.
COMMON_MSG_NOT_WRITEABLE_ERROR variable	You cannot write to a message as it is in read-only mode.
COMMON_MSG_RETRIEVE_ERROR variable	Error retrieving a message from the client message store.
COMMON_MSG_STORE_ERROR variable	Error storing a message in the client message store.
COMMON_MSG_STORE_NOT_INITIALIZED variable	The message store has not been initialized for messaging.
COMMON_MSG_STORE_TOO_LARGE variable	The message store is too large relative to the free disk space on the device.
COMMON_NO_DEST_ERROR variable	No destination.
COMMON_NO_IMPLEMENTATION variable	The function is not implemented.
COMMON_NOT_OPEN_ERROR variable	The QAManager is not open.

Name	Description
COMMON_OPEN_ERROR variable	Error opening a connection to the message store.
COMMON_OPEN_LOG_FILE_ERROR variable	Error opening the log file.
COMMON_OPEN_MAXTHREADS_ERROR variable	Cannot open the QAManager because the maximum number of concurrent server requests is not high enough.
COMMON_SELECTOR_SYNTAX_ERROR variable	The given selector has a syntax error.
COMMON_SET_PROPERTY_ERROR variable	Error setting the property.
COMMON_TERMINATE_ERROR variable	Termination error.
COMMON_UNEXPECTED_EOM_ERROR variable	Unexpected end of message reached.
COMMON_UNREPRESENTABLE_TIMESTAMP variable	The timestamp is outside of the acceptable range.
QA_NO_ERROR variable	No error.

Remarks

A QAError object is used internally by the QAManager object to keep track of errors associated with messaging operations. The application programmer should not need to create an instance of this class. The error constants should be used by the application programmer to interpret error codes returned by QAManager::getLastError

See also

- "QAManagerBase.getLastErrorMsg method [QAnywhere C++]" on page 408

Example

Assume your QAManager instance is called qa_mgr. The following example, uses the QAManager::getLastError method to compare the last error code to QAError::QA_NO_ERROR.

```
if (qa_mgr->getLastError() != QAError::QA_NO_ERROR) {
    // Process error.
}
```

COMMON_ALREADY_OPEN_ERROR variable

The QAManager is already open.

Syntax

```
public static const qa_int COMMON_ALREADY_OPEN_ERROR;
```


COMMON_GET_INIT_FILE_ERROR variable

Unable to access the client properties file.

Syntax

```
public static const qa_int COMMON_GET_INIT_FILE_ERROR;
```

COMMON_GET_PROPERTY_ERROR variable

Error retrieving the property.

Syntax

```
public static const qa_int COMMON_GET_PROPERTY_ERROR;
```

COMMON_GETQUEUEDEPTH_ERROR variable

Error getting queue depth.

Syntax

```
public static const qa_int COMMON_GETQUEUEDEPTH_ERROR;
```

COMMON_GETQUEUEDEPTH_ERROR_INVALID_ARG variable

Cannot use QAManagerBase.getQueueDepth on a given destination when filter is ALL.

Syntax

```
public static const qa_int COMMON_GETQUEUEDEPTH_ERROR_INVALID_ARG;
```

COMMON_GETQUEUEDEPTH_ERROR_NO_STORE_ID variable

Cannot use QAManagerBase.getQueueDepth when the message store ID has not been set.

Syntax

```
public static const qa_int COMMON_GETQUEUEDEPTH_ERROR_NO_STORE_ID;
```

COMMON_INIT_ERROR variable

Initialization error.

Syntax

```
public static const qa_int COMMON_INIT_ERROR;
```

COMMON_INIT_THREAD_ERROR variable

Error initializing the background thread.

Syntax

```
public static const qa_int COMMON_INIT_THREAD_ERROR;
```

COMMON_INVALID_PROPERTY variable

There is an invalid property in the client properties file.

Syntax

```
public static const qa_int COMMON_INVALID_PROPERTY;
```

COMMON_MSG_ACKNOWLEDGE_ERROR variable

Error acknowledging the message.

Syntax

```
public static const qa_int COMMON_MSG_ACKNOWLEDGE_ERROR;
```

COMMON_MSG_CANCEL_ERROR variable

Error canceling message.

Syntax

```
public static const qa_int COMMON_MSG_CANCEL_ERROR;
```

COMMON_MSG_CANCEL_ERROR_SENT variable

Error canceling message.

Syntax

```
public static const qa_int COMMON_MSG_CANCEL_ERROR_SENT;
```

Remarks

Cannot cancel a message that has already been sent.

COMMON_MSG_NOT_WRITEABLE_ERROR variable

You cannot write to a message as it is in read-only mode.

Syntax

```
public static const qa_int COMMON_MSG_NOT_WRITEABLE_ERROR;
```

COMMON_MSG_RETRIEVE_ERROR variable

Error retrieving a message from the client message store.

Syntax

```
public static const qa_int COMMON_MSG_RETRIEVE_ERROR;
```

COMMON_MSG_STORE_ERROR variable

Error storing a message in the client message store.

Syntax

```
public static const qa_int COMMON_MSG_STORE_ERROR;
```

COMMON_MSG_STORE_NOT_INITIALIZED variable

The message store has not been initialized for messaging.

Syntax

```
public static const qa_int COMMON_MSG_STORE_NOT_INITIALIZED;
```

COMMON_MSG_STORE_TOO_LARGE variable

The message store is too large relative to the free disk space on the device.

Syntax

```
public static const qa_int COMMON_MSG_STORE_TOO_LARGE;
```

COMMON_NO_DEST_ERROR variable

No destination.

Syntax

```
public static const qa_int COMMON_NO_DEST_ERROR;
```

COMMON_NO_IMPLEMENTATION variable

The function is not implemented.

Syntax

```
public static const qa_int COMMON_NO_IMPLEMENTATION;
```

COMMON_NOT_OPEN_ERROR variable

The QAManager is not open.

Syntax

```
public static const qa_int COMMON_NOT_OPEN_ERROR;
```

COMMON_OPEN_ERROR variable

Error opening a connection to the message store.

Syntax

```
public static const qa_int COMMON_OPEN_ERROR;
```

COMMON_OPEN_LOG_FILE_ERROR variable

Error opening the log file.

Syntax

```
public static const qa_int COMMON_OPEN_LOG_FILE_ERROR;
```

COMMON_OPEN_MAXTHREADS_ERROR variable

Cannot open the QAManager because the maximum number of concurrent server requests is not high enough.

Syntax

```
public static const qa_int COMMON_OPEN_MAXTHREADS_ERROR;
```


Remarks

For more information, see "-gn dbsrv12 server option" [*SQL Anywhere Server - Database Administration*].

COMMON_SELECTOR_SYNTAX_ERROR variable

The given selector has a syntax error.

Syntax

```
public static const qa_int COMMON_SELECTOR_SYNTAX_ERROR;
```

COMMON_SET_PROPERTY_ERROR variable

Error setting the property.

Syntax

```
public static const qa_int COMMON_SET_PROPERTY_ERROR;
```

COMMON_TERMINATE_ERROR variable

Termination error.

Syntax

```
public static const qa_int COMMON_TERMINATE_ERROR;
```

COMMON_UNEXPECTED_EOM_ERROR variable

Unexpected end of message reached.

Syntax

```
public static const qa_int COMMON_UNEXPECTED_EOM_ERROR;
```

COMMON_UNREPRESENTABLE_TIMESTAMP variable

The timestamp is outside of the acceptable range.

Syntax

```
public static const qa_int COMMON_UNREPRESENTABLE_TIMESTAMP;
```

QA_NO_ERROR variable

No error.

Syntax

```
public static const qa_int QA_NO_ERROR;
```

QAManager class

The QAManager class derives from QAManagerBase and manages non-transactional QAnywhere messaging operations.

Syntax

```
public class QAManager : QAManagerBase
```

Base classes

- "QAManagerBase class [QAnywhere C++]" on page 394

Members

All members of QAManager class, including all inherited members.

Name	Description
acknowledge method	Acknowledges that the client application successfully received a QAnywhere message.
acknowledgeAll method	Acknowledges that the client application successfully received all unacknowledged QAnywhere messages.
acknowledgeUntil method	Acknowledges the given QAMessage instance and all unacknowledged messages received before the given message.
beginEnumStoreProperty-Names method	Begins an enumeration of message store property names.
browseClose method	Frees the resources associated with a browse operation.
browseMessages method	Begins a browse of messages queued in the message store.
browseMessagesByID method	Begins a browse of the message that is queued in the message store, with the given message ID.
browseMessagesByQueue method	Begins a browse of messages queued in the message store for the given queue.

Name	Description
browseMessagesBySelector method	Begins a browse of messages queued in the message store that satisfy the given selector.
browseNextMessage method	Returns the next message for the given browse operation, returning null if there are no more messages.
cancelMessage method	Cancels the message with the given message ID.
close method	Closes the connection to the QAnywhere message system and releases any resources used by the QAManagerBase.
createBinaryMessage method	Creates a QABinaryMessage instance.
createTextMessage method	Creates a QATextMessage instance.
deleteMessage method	Deletes a QAMessage object.
endEnumStorePropertyNames method	Frees the resources associated with a message store property name enumeration.
getAllQueueDepth method	Returns the total depth of all queues, based on a given filter.
getBooleanStoreProperty method	Gets a boolean value for a predefined or custom message store property.
getByteStoreProperty method	Gets a byte value for a predefined or custom message store property.
getDoubleStoreProperty method	Gets a double value for a predefined or custom message store property.
getFloatStoreProperty method	Gets a float value for a predefined or custom message store property.
getIntStoreProperty method	Gets an int value for a predefined or custom message store property.
getLastError method	The error code associated with the last executed QAManagerBase method.
getLastErrorMsg method	The error text associated with the last executed QAManagerBase method.
getLastNativeError method	The native error code associated with the last executed QAManagerBase method.

Name	Description
getLongStoreProperty method	Gets a long value for a predefined or custom message store property.
getMessage method	Returns the next available QAMessage sent to the specified address.
getMessageBySelector method	Returns the next available QAMessage sent to the specified address that satisfies the given selector.
getMessageBySelectorNoWait method	Returns the next available QAMessage sent to the given address that satisfies the given selector.
getMessageBySelectorTimeout method	Returns the next available QAMessage sent to the given address that satisfies the given selector.
getMessageNoWait method	Returns the next available QAMessage sent to the given address.
getMessageTimeout method	Returns the next available QAMessage sent to the given address.
getMode method	Returns the QAManager acknowledgement mode for received messages.
getQueueDepth method	Returns the depth of a queue, based on a given filter.
getShortStoreProperty method	Gets a short value for a predefined or custom message store property.
getStringStoreProperty method	Gets a string value for a predefined or custom message store property.
nextStorePropertyName method	Returns the message store property name for the given enumeration.
open method	Opens the QAManager with the given AcknowledgementMode value.
putMessage method	Puts a message into the queue for the given destination.
putMessageTimeToLive method	Puts a message into the queue for the given destination and a given time-to-live in milliseconds.
recover method	Force all unacknowledged messages into a state of unreceived.
setBooleanStoreProperty method	Sets a predefined or custom message store property to a boolean value.
setByteStoreProperty method	Sets a predefined or custom message store property to a byte value.

Name	Description
setDoubleStoreProperty method	Sets a predefined or custom message store property to a double value.
setFloatStoreProperty method	Sets a predefined or custom message store property to a float value.
setIntStoreProperty method	Sets a predefined or custom message store property to an int value.
setLongStoreProperty method	Sets a predefined or custom message store property to a long value.
setMessageListener method	Sets a message listener class to receive QAnywhere messages asynchronously.
setMessageListenerBySelector method	Sets a message listener class to receive QAnywhere messages asynchronously, with a message selector.
setProperty method	Allows you to set QAnywhere manager configuration properties programmatically.
setShortStoreProperty method	Sets a predefined or custom message store property to a short value.
setStringStoreProperty method	Sets a predefined or custom message store property to a string value.
start method	Starts the QAManagerBase for receiving incoming messages in message listeners.
stop method	Stops the QAManagerBase's reception of incoming messages.
triggerSendReceive method	Causes a synchronization with the QAnywhere message server, uploading any messages addressed to other clients, and downloading any messages addressed to the local client.

Remarks

For a detailed description of derived behavior, see QAManagerBase.

The QAManager can be configured for implicit or explicit acknowledgement as defined in the AcknowledgementMode enumeration. To acknowledge messages as part of a transaction, use QATransactionalManager.

Use the QAManagerFactory to create QAManager and QATransactionalManager objects.

See also

- "QAManagerFactory class [QAnywhere C++]" on page 425
- "QAManagerBase class [QAnywhere C++]" on page 394
- "QATransactionalManager class [QAnywhere C++]" on page 457
- "AcknowledgementMode class [QAnywhere C++]" on page 354

acknowledge method

Acknowledges that the client application successfully received a QAnywhere message.

Syntax

```
public virtual qa_bool acknowledge(QAMessage * msg)
```

Parameters

- **msg** The message to acknowledge.

Returns

True if and only if the operation succeeded.

Remarks

Note
When a QAMessage is acknowledged, its MessageProperties::STATUS property changes to StatusCodes.RECEIVED. When a QAMessage status changes to StatusCodes.RECEIVED, it can be deleted using the default delete rule.

For more information about delete rules, see "Message delete rules" on page 741.

See also

- "QAManager.acknowledgeAll method [QAnywhere C++]" on page 392
- "QAManager.acknowledgeUntil method [QAnywhere C++]" on page 393

acknowledgeAll method

Acknowledges that the client application successfully received all unacknowledged QAnywhere messages.

Syntax

```
public virtual qa_bool acknowledgeAll()
```

Returns

True if and only if the operation succeeded.

Remarks

Note
When a QAMessage is acknowledged, its MessageProperties::STATUS property changes to StatusCodes.RECEIVED. When a QAMessage status changes to StatusCodes.RECEIVED, it can be deleted using the default delete rule.

For more information about delete rules, see "Message delete rules" on page 741.

See also

- "QAManager.acknowledge method [QAnywhere C++]" on page 392
- "QAManager.acknowledgeUntil method [QAnywhere C++]" on page 393

acknowledgeUntil method

Acknowledges the given QAMessage instance and all unacknowledged messages received before the given message.

Syntax

```
public virtual qa_bool acknowledgeUntil(QAMessage * msg)
```

Parameters

- **msg** The last message to acknowledge. All earlier unacknowledged messages are also acknowledged.

Returns

True if and only if the operation succeeded.

Remarks

Note
When a QAMessage is acknowledged, its MessageProperties::STATUS property changes to StatusCodes.RECEIVED. When a QAMessage status changes to StatusCodes.RECEIVED, it can be deleted using the default delete rule.

For more information about delete rules, see "Message delete rules" on page 741.

See also

- "QAManager.acknowledge method [QAnywhere C++]" on page 392
- "QAManager.acknowledgeAll method [QAnywhere C++]" on page 392

open method

Opens the QAManager with the given AcknowledgementMode value.

Syntax

```
public virtual qa_bool open(qa_short mode)
```

Parameters

- **mode** The acknowledgement mode.

Returns

True if and only if the operation succeeded.

Remarks

The open method must be the first method called after creating a QAManager.

If a database connection error is detected, you can re-open a QAManager by calling the close function followed by the open function. When re-opening a QAManager, you do not need to re-create it, reset the properties, or reset the message listeners. The properties of the QAManager cannot be changed after the first open, and subsequent open calls must supply the same acknowledgement mode.

See also

- "AcknowledgementMode class [QAnywhere C++]" on page 354

recover method

Force all unacknowledged messages into a state of unreceived.

Syntax

```
public virtual qa_bool recover()
```

Returns

True if and only if the operation succeeded.

Remarks

That is, these messages must be received again using QAManager::getMessage().

QAManagerBase class

This class acts as a base class for QATransactionalManager and QAManager, which manage transactional and non-transactional messaging, respectively.

Syntax

```
public class QAManagerBase
```


Derived classes

- "QAManager class [QAnywhere C++]" on page 388
- "QATransactionalManager class [QAnywhere C++]" on page 457

Members

All members of QAManagerBase class, including all inherited members.

Name	Description
beginEnumStoreProperty-Names method	Begins an enumeration of message store property names.
browseClose method	Frees the resources associated with a browse operation.
browseMessages method	Begins a browse of messages queued in the message store.
browseMessagesByID meth-od	Begins a browse of the message that is queued in the message store, with the given message ID.
browseMessagesByQueue method	Begins a browse of messages queued in the message store for the given queue.
browseMessagesBySelector method	Begins a browse of messages queued in the message store that satisfy the given selector.
browseNextMessage method	Returns the next message for the given browse operation, returning null if there are no more messages.
cancelMessage method	Cancels the message with the given message ID.
close method	Closes the connection to the QAnywhere message system and releases any resources used by the QAManagerBase.
createBinaryMessage meth-od	Creates a QABinaryMessage instance.
createTextMessage method	Creates a QATextMessage instance.
deleteMessage method	Deletes a QAMessage object.
endEnumStoreProperty-Names method	Frees the resources associated with a message store property name enumeration.
getAllQueueDepth method	Returns the total depth of all queues, based on a given filter.
getBooleanStoreProperty method	Gets a boolean value for a predefined or custom message store proper-ty.

Name	Description
getByteStoreProperty method	Gets a byte value for a predefined or custom message store property.
getDoubleStoreProperty method	Gets a double value for a predefined or custom message store property.
getFloatStoreProperty method	Gets a float value for a predefined or custom message store property.
getIntStoreProperty method	Gets an int value for a predefined or custom message store property.
getLastError method	The error code associated with the last executed QAManagerBase method.
getLastErrorMsg method	The error text associated with the last executed QAManagerBase method.
getLastNativeError method	The native error code associated with the last executed QAManagerBase method.
getLongStoreProperty method	Gets a long value for a predefined or custom message store property.
getMessage method	Returns the next available QAMessage sent to the specified address.
getMessageBySelector method	Returns the next available QAMessage sent to the specified address that satisfies the given selector.
getMessageBySelectorNoWait method	Returns the next available QAMessage sent to the given address that satisfies the given selector.
getMessageBySelectorTimeout method	Returns the next available QAMessage sent to the given address that satisfies the given selector.
getMessageNoWait method	Returns the next available QAMessage sent to the given address.
getMessageTimeout method	Returns the next available QAMessage sent to the given address.
getMode method	Returns the QAManager acknowledgement mode for received messages.
getQueueDepth method	Returns the depth of a queue, based on a given filter.
getShortStoreProperty method	Gets a short value for a predefined or custom message store property.

Name	Description
getStringStoreProperty method	Gets a string value for a predefined or custom message store property.
nextStorePropertyName method	Returns the message store property name for the given enumeration.
putMessage method	Puts a message into the queue for the given destination.
putMessageTimeToLive method	Puts a message into the queue for the given destination and a given time-to-live in milliseconds.
setBooleanStoreProperty method	Sets a predefined or custom message store property to a boolean value.
setByteStoreProperty method	Sets a predefined or custom message store property to a byte value.
setDoubleStoreProperty method	Sets a predefined or custom message store property to a double value.
setFloatStoreProperty method	Sets a predefined or custom message store property to a float value.
setIntStoreProperty method	Sets a predefined or custom message store property to an int value.
setLongStoreProperty method	Sets a predefined or custom message store property to a long value.
setMessageListener method	Sets a message listener class to receive QAnywhere messages asynchronously.
setMessageListenerBySelector method	Sets a message listener class to receive QAnywhere messages asynchronously, with a message selector.
setProperty method	Allows you to set QAnywhere manager configuration properties programmatically.
setShortStoreProperty method	Sets a predefined or custom message store property to a short value.
setStringStoreProperty method	Sets a predefined or custom message store property to a string value.
start method	Starts the QAManagerBase for receiving incoming messages in message listeners.
stop method	Stops the QAManagerBase's reception of incoming messages.

Name	Description
triggerSendReceive method	Causes a synchronization with the QAnywhere message server, uploading any messages addressed to other clients, and downloading any messages addressed to the local client.

Remarks

Use the QAManagerBase::start() method to allow a QAManagerBase instance to listen for messages. There must be only a single instance of QAManagerBase per thread in your application.

You can use instances of this class to create and manage QAnywhere messages. Use the QAManagerBase::createBinaryMessage() method and the QAManagerBase::createTextMessage() method to create appropriate QAMessage instances. QAMessage instances provide a variety of methods to set message content and properties. To send QAnywhere messages, use the QAManager::putMessage() to place the addressed message in the local message store queue. The message is transmitted by the QAnywhere Agent based on its transmission policies or when you call the QAManagerBase::triggerSendReceive().

For more information about qaagent transmission policies, see "Determining when message transmission should occur on the client" on page 46.

Messages are released from memory when you close a QAManagerBase instance using the QAManagerBase::close().

You can use QAManagerBase::getLastError, QAManagerBase::getLastNativeError, and QAManagerBase::getLastErrorMessage to return error information when a QAException occurs. QAManagerBase also provides methods to set and get message store properties.

For more information, see "Client message store properties" on page 26 and the MessageStoreProperties.

See also

- "QATransactionalManager class [QAnywhere C++]" on page 457
- "QAManager class [QAnywhere C++]" on page 388

beginEnumStorePropertyNames method

Begins an enumeration of message store property names.

Syntax

```
public virtual qa_store_property_enum_handle
beginEnumStorePropertyNames()
```

Returns

A handle that is supplied to QAManagerBase::nextStorePropertyName.

Remarks

The handle returned by this method is supplied to the QAManagerBase::nextStorePropertyName. This method and the QAManagerBase::nextStorePropertyName can be used to enumerate the message store property names at the time this method was called. Message store properties cannot be set between the QAManagerBase::beginEnumStorePropertyNames and the QAManagerBase::endEnumStorePropertyNames calls.

See also

- "QAManagerBase.nextStorePropertyName method [QAnywhere C++]" on page 415
- "QAManagerBase.beginEnumStorePropertyNames method [QAnywhere C++]" on page 398
- "QAManagerBase.endEnumStorePropertyNames method [QAnywhere C++]" on page 404

browseClose method

Frees the resources associated with a browse operation.

Syntax

```
public virtual void browseClose(qa_browse_handle handle)
```

Parameters

- **handle** A handle returned by one of the begin browse operations.

browseMessages method

Begins a browse of messages queued in the message store.

Syntax

```
public virtual qa_browse_handle browseMessages()
```

Returns

A handle that is supplied to QAManagerBase::browseNextMessage

Remarks

The handle returned by this method is supplied to QAManagerBase::browseNextMessage. This method and the QAManagerBase::browseNextMessage can be used to enumerate the messages in the message store at the time this method was called.

The messages are just being browsed, so they cannot be acknowledged. Use QAManagerBase::getMessage to receive messages so they can be acknowledged.

See also

- "QAManagerBase.browseNextMessage method [QAnywhere C++]" on page 402
- "QAManagerBase.browseMessagesByQueue method [QAnywhere C++]" on page 400
- "QAManagerBase.browseMessagesByID method [QAnywhere C++]" on page 400
- "QAManagerBase.browseClose method [QAnywhere C++]" on page 399

browseMessagesByID method

Begins a browse of the message that is queued in the message store, with the given message ID.

Syntax

```
public virtual qa_browse_handle browseMessagesByID(
     qa_const_string msgid
)
```

Parameters

- **msgid** The message ID.

Returns

A handle that is supplied to browseNextMessage.

Remarks

The handle returned by this method is supplied to QAManagerBase::browseNextMessage. This method and QAManagerBase::browseNextMessage can be used to enumerate the messages in the message store at the time this method was called.

The messages are just being browsed, so they cannot be acknowledged. Use QAManagerBase::getMessage to receive messages so they can be acknowledged.

See also

- "QAManagerBase.browseNextMessage method [QAnywhere C++]" on page 402
- "QAManagerBase.browseMessagesByQueue method [QAnywhere C++]" on page 400
- "QAManagerBase.browseMessages method [QAnywhere C++]" on page 399
- "QAManagerBase.browseClose method [QAnywhere C++]" on page 399

browseMessagesByQueue method

Begins a browse of messages queued in the message store for the given queue.

Syntax

```
public virtual qa_browse_handle browseMessagesByQueue(
     qa_const_string address
)
```


Parameters

- **address** The queue in which to browse.

Returns

A handle that is supplied to browseNextMessage.

Remarks

The handle returned by this method is supplied to QAManagerBase::browseNextMessage. This method and QAManagerBase::browseNextMessage can be used to enumerate the messages in the message store at the time this method was called.

The messages are just being browsed, so they cannot be acknowledged. Use QAManagerBase::getMessage to receive messages so they can be acknowledged.

See also

- "QAManagerBase.browseNextMessage method [QAnywhere C++]" on page 402
- "QAManagerBase.browseMessagesByID method [QAnywhere C++]" on page 400
- "QAManagerBase.browseMessages method [QAnywhere C++]" on page 399
- "QAManagerBase.browseClose method [QAnywhere C++]" on page 399

browseMessagesBySelector method

Begins a browse of messages queued in the message store that satisfy the given selector.

Syntax

```
public virtual qa_browse_handle browseMessagesBySelector(
     qa_const_string selector
)
```

Parameters

- **selector** The selector.

Returns

A handle that is supplied to browseNextMessage.

Remarks

The handle returned by this method is supplied to QAManagerBase::browseNextMessage. This method and QAManagerBase::browseNextMessage can be used to enumerate the messages in the message store at the time this method was called.

The messages are just being browsed, so they cannot be acknowledged.

Use QAManagerBase::getMessage to receive messages so they can be acknowledged.

See also

- "QAManagerBase.browseNextMessage method [QAnywhere C++]" on page 402
- "QAManagerBase.browseMessagesByID method [QAnywhere C++]" on page 400
- "QAManagerBase.browseMessagesByQueue method [QAnywhere C++]" on page 400
- "QAManagerBase.browseMessages method [QAnywhere C++]" on page 399
- "QAManagerBase.browseClose method [QAnywhere C++]" on page 399

browseNextMessage method

Returns the next message for the given browse operation, returning null if there are no more messages.

Syntax

```
public virtual QAMessage * browseNextMessage(qa_browse_handle handle)
```

Parameters

- **handle** A handle returned by one of the begin browse operations.

Returns

The next message, or qa_null if there are no more messages.

Remarks

To obtain the handle to browsed messages, use QAManagerBase::browseMessages or other QAManagerBase methods which allow you to browse messages by queue or message ID.

See also

- "QAManagerBase.browseMessages method [QAnywhere C++]" on page 399
- "QAManagerBase.browseMessagesByQueue method [QAnywhere C++]" on page 400
- "QAManagerBase.browseMessagesByID method [QAnywhere C++]" on page 400
- "QAManagerBase.browseClose method [QAnywhere C++]" on page 399

cancelMessage method

Cancels the message with the given message ID.

Syntax

```
public virtual qa_bool cancelMessage(qa_const_string msgid)
```

Parameters

- **msgid** The ID of the message to cancel.

Returns

True if and only if the operation succeeded.

Remarks

The cancelMessage method puts a message into a canceled state before it is transmitted. With the default delete rules of the QAnywhere Agent, canceled messages are eventually deleted from the message store.

The cancelMessage method fails if the message is already in a final state, or if it has been transmitted to the central messaging server.

For more information about delete rules, see "Message delete rules" on page 741.

close method

Closes the connection to the QAnywhere message system and releases any resources used by the QAManagerBase.

Syntax

```
public virtual qa_bool close()
```

Returns

True if and only if the operation succeeded.

Remarks

Subsequent calls to close() are ignored. When an instance of QAManagerBase is closed, it cannot be re-opened; you must create and open a new QAManagerBase instance in this case.

If a database connection error is detected, you can re-open a QAManager by calling the close method followed by the open method. When re-opening a QAManager, you do not need to re-create it, reset the properties, or reset the message listeners. The properties of the QAManager cannot be changed after the first open, and subsequent open calls must supply the same acknowledgement mode.

See also

- "QAManager.open method [QAnywhere C++]" on page 393
- "QATransactionalManager.open method [QAnywhere C++]" on page 462

createBinaryMessage method

Creates a QABinaryMessage instance.

Syntax

```
public virtual QABinaryMessage * createBinaryMessage()
```

Returns

A new QABinaryMessage instance.

Remarks

A QABinaryMessage instance is used to send a message containing a message body of uninterpreted bytes.

See also

- "QABinaryMessage class [QAnywhere C++]" on page 367

createTextMessage method

Creates a QATextMessage instance.

Syntax

```
public virtual QATextMessage * createTextMessage()
```

Returns

A new QATextMessage instance.

Remarks

A QATextMessage object is used to send a message containing a string message body.

See also

- "QATextMessage class [QAnywhere C++]" on page 452

deleteMessage method

Deletes a QAMessage object.

Syntax

```
public virtual void deleteMessage(QAMessage * msg)
```

Parameters

- **msg** The message to delete.

Remarks

By default, messages created by QAManagerBase::createTextMessage or QAManagerBase::createBinaryMessage are deleted automatically when the QAManagerBase is closed. This method allows more control over when messages are deleted.

endEnumStorePropertyNames method

Frees the resources associated with a message store property name enumeration.

Syntax

```
public virtual void endEnumStorePropertyNames(
     qa_store_property_enum_handle h
)
```

Parameters

- **h** A handle returned by beginEnumStorePropertyNames.

See also

- "QAManagerBase.beginEnumStorePropertyNames method [QAnywhere C++]" on page 398

getAllQueueDepth method

Returns the total depth of all queues, based on a given filter.

Syntax

```
public virtual qa_int getAllQueueDepth(qa_short filter)
```

Parameters

- **filter** A filter indicating incoming messages, outgoing messages, or all messages.

Returns

The number of messages, or -1 if an error occurs.

Remarks

The incoming depth of a queue is the number of incoming messages which have not been received (for example, using QAManagerBase::getMessage). The outgoing depth of a queue is the number of outgoing messages (including uncommitted) that have not been transmitted to the server.

See also

- "QueueDepthFilter class [QAnywhere C++]" on page 462

getBooleanStoreProperty method

Gets a boolean value for a predefined or custom message store property.

Syntax

```
public virtual qa_bool getBooleanStoreProperty(
     qa_const_string name,
    qa_bool * value
)
```

Parameters

- **name** The predefined or custom property name.

- **value** The destination for the boolean value.

Returns

True if and only if the operation succeeded.

Remarks

You can use this method to access predefined or user-defined client store properties.

For a list of predefined properties, see MessageStoreProperties.

For more information, see "Client message store properties" on page 26.

getByteStoreProperty method

Gets a byte value for a predefined or custom message store property.

Syntax

```
public virtual qa_bool getByteStoreProperty(
     qa_const_string name,
    qa_byte * value
)
```

Parameters

- **name** The predefined or custom property name.
- **value** The destination for the byte value.

Returns

True if and only if the operation succeeded.

Remarks

You can use this method to access predefined or user-defined client store properties.

For a list of predefined properties, see MessageStoreProperties.

For more information, see "Client message store properties" on page 26.

getDoubleStoreProperty method

Gets a double value for a predefined or custom message store property.

Syntax

```
public virtual qa_bool getDoubleStoreProperty(
     qa_const_string name,
    qa_double * value
)
```


Parameters

- **name** The predefined or custom property name.
- **value** The destination for the double value.

Returns

True if and only if the operation succeeded.

Remarks

You can use this method to access predefined or user-defined client store properties.

For a list of predefined properties, see MessageStoreProperties.

For more information, see "Client message store properties" on page 26.

getFloatStoreProperty method

Gets a float value for a predefined or custom message store property.

Syntax

```
public virtual qa_bool getFloatStoreProperty(
     qa_const_string name,
    qa_float * value
)
```

Parameters

- **name** The predefined or custom property name.
- **value** The destination for the float value.

Returns

True if and only if the operation succeeded.

Remarks

You can use this method to access predefined or user-defined client store properties.

For a list of predefined properties, see MessageStoreProperties.

For more information, see "Client message store properties" on page 26.

getIntStoreProperty method

Gets an int value for a predefined or custom message store property.

Syntax

```
public virtual qa_bool getIntStoreProperty(
     qa_const_string name,
    qa_int * value
)
```

Parameters

- **name** The predefined or custom property name.
- **value** The destination for the int value.

Returns

True if and only if the operation succeeded.

Remarks

You can use this method to access predefined or user-defined client store properties.

For a list of predefined properties, see MessageStoreProperties.

For more information, see "Client message store properties" on page 26.

getLastError method

The error code associated with the last executed QAManagerBase method.

Syntax

```
public virtual qa_int getLastError()
```

Returns

The error code.

Remarks

0 indicates no error.

See also

- "QAManagerBase.getLastNativeError method [QAnywhere C++]" on page 409
- "QAManagerBase.getLastErrorMsg method [QAnywhere C++]" on page 408
- "QAError class [QAnywhere C++]" on page 380

getLastErrorMsg method

The error text associated with the last executed QAManagerBase method.

Syntax

```
public virtual qa_string getLastErrorMsg()
```


Returns

The error message.

Remarks

This method returns null if QAManagerBase::getLastError returns 0. You can retrieve this property after catching a QAError.

See also

- "QAManagerBase.getLastError method [QAnywhere C++]" on page 408
- "QAManagerBase.getLastNativeError method [QAnywhere C++]" on page 409
- "QAError class [QAnywhere C++]" on page 380

getLastNativeError method

The native error code associated with the last executed QAManagerBase method.

Syntax

```
public virtual an_sql_code getLastNativeError()
```

Returns

The native error code.

Remarks

0 indicates no error.

See also

- "QAManagerBase.getLastError method [QAnywhere C++]" on page 408
- "QAManagerBase.getLastErrorMsg method [QAnywhere C++]" on page 408
- "QAError class [QAnywhere C++]" on page 380

getLongStoreProperty method

Gets a long value for a predefined or custom message store property.

Syntax

```
public virtual qa_bool getLongStoreProperty(
     qa_const_string name,
    qa_long * value
)
```

Parameters

- **name** The predefined or custom property name.
- **value** The destination for the long value.

Returns

True if and only if the operation succeeded.

Remarks

You can use this method to access predefined or user-defined client store properties.

For a list of predefined properties, see MessageStoreProperties.

For more information, see "Client message store properties" on page 26.

getMessage method

Returns the next available QAMessage sent to the specified address.

Syntax

```
public virtual QAMessage * getMessage(qa_const_string address)
```

Parameters

- **address** The destination.

Returns

The next QAMessage, or null if no message is available.

Remarks

The address parameter specifies a local queue name. The address can be in the form 'store-id\queue-name' or 'queue-name'. If there is no message available, this call blocks indefinitely until a message is available. Use this method to receive messages synchronously.

For more information about receiving messages asynchronously (using a message event handler), see "Receiving messages asynchronously" on page 69.

getMessageBySelector method

Returns the next available QAMessage sent to the specified address that satisfies the given selector.

Syntax

```
public virtual QAMessage * getMessageBySelector(
     qa_const_string address,
    qa_const_string selector
)
```

Parameters

- **address** The destination.
- **selector** The selector.

Returns

The next QAMessage, or null if no message is available.

Remarks

The address parameter specifies a local queue name. The address can be in the form 'store-id\queue-name' or 'queue-name'. If there is no message available, this call blocks indefinitely until a message is available. Use this method to receive messages synchronously.

For more information about receiving messages asynchronously (using a message event handler), see "Receiving messages asynchronously" on page 69.

getMessageBySelectorNoWait method

Returns the next available QAMessage sent to the given address that satisfies the given selector.

Syntax

```
public virtual QAMessage * getMessageBySelectorNoWait(
     qa_const_string address,
    qa_const_string selector
)
```

Parameters

- **address** The destination.
- **selector** The selector.

Returns

The next message, or qa_null if no message is available.

Remarks

The address parameter specifies a local queue name. The address can be in the form 'store-id\queue-name' or 'queue-name'. If no message is available, this method returns immediately. Use this method to receive messages synchronously.

For more information about receiving messages asynchronously (using a message event handler), see "Receiving messages asynchronously" on page 69.

getMessageBySelectorTimeout method

Returns the next available QAMessage sent to the given address that satisfies the given selector.

Syntax

```
public virtual QAMessage * getMessageBySelectorTimeout(
     qa_const_string address,
    qa_const_string selector,
```

```
    qa_long timeout
)
```

Parameters

- **address** The destination.
- **selector** The selector.
- **timeout** the maximum time, in milliseconds, to wait

Returns

The next QAMessage, or null if no message is available.

Remarks

The address parameter specifies a local queue name. The address can be in the form 'store-id\queue-name' or 'queue-name'. If no message is available, this method waits for the specified timeout and then returns. Use this method to receive messages synchronously.

For more information about receiving messages asynchronously (using a message event handler), see "Receiving messages asynchronously" on page 69.

getMessageNoWait method

Returns the next available QAMessage sent to the given address.

Syntax

```
public virtual QAMessage * getMessageNoWait(qa_const_string address)
```

Parameters

- **address** The destination.

Returns

The next message, or qa_null if no message is available.

Remarks

The address parameter specifies a local queue name. The address can be in the form 'store-id\queue-name' or 'queue-name'. If no message is available, this method returns immediately. Use this method to receive messages synchronously.

For more information about receiving messages asynchronously (using a message event handler), see "Receiving messages asynchronously" on page 69.

getMessageTimeout method

Returns the next available QAMessage sent to the given address.

Syntax

```
public virtual QAMessage * getMessageTimeout(
     qa_const_string address,
    qa_long timeout
)
```

Parameters

- **address** The destination
- **timeout** The maximum time, in milliseconds, to wait

Returns

The next QAMessage, or null if no message is available.

Remarks

The address parameter specifies a local queue name. The address can be in the form 'store-id\queue-name' or 'queue-name'. If no message is available, this method waits for the specified timeout and then returns. Use this method to receive messages synchronously.

For more information about receiving messages asynchronously (using a message event handler), see "Receiving messages asynchronously" on page 69.

getMode method

Returns the QAManager acknowledgement mode for received messages.

Syntax

```
public virtual qa_short getMode()
```

Returns

The acknowledgement mode.

Remarks

For a list of values, see the AcknowledgementMode class.

AcknowledgementMode::EXPLICIT_ACKNOWLEDGEMENT and AcknowledgementMode::IMPLICIT_ACKNOWLEDGEMENT apply to QAManager instances; AcknowledgementMode::TRANSACTIONAL is the mode for QATransactionalManager instances.

See also

- "AcknowledgementMode class [QAnywhere C++]" on page 354

getQueueDepth method

Returns the depth of a queue, based on a given filter.

Syntax

```
public virtual qa_int getQueueDepth(
     qa_const_string address,
    qa_short filter
)
```

Parameters

- **address** The queue name.
- **filter** A filter indicating incoming messages, outgoing messages, or all messages.

Returns

The number of messages in the queue, or -1 if an error occurs.

Remarks

The incoming depth of the queue is the number of incoming messages which have not been received (for example, using QAManagerBase::getMessage). The outgoing depth of a queue is the number of outgoing messages (including uncommitted) that have not been transmitted to the server.

If getQueueDepth is called with the LOCAL filter and a queue is specified, it returns the number of unreceived local messages that are addressed to that queue. If a queue is not specified, it returns the total number of unreceived local messages in the message store, excluding system messages.

See also

- "QueueDepthFilter class [QAnywhere C++]" on page 462

getShortStoreProperty method

Gets a short value for a predefined or custom message store property.

Syntax

```
public virtual qa_bool getShortStoreProperty(
     qa_const_string name,
    qa_short * value
)
```

Parameters

- **name** The predefined or custom property name.
- **value** The destination for the short value.

Returns

True if and only if the operation succeeded.

Remarks

You can use this method to access predefined or user-defined client store properties.

For a list of predefined properties, see MessageStoreProperties.

For more information, see "Client message store properties" on page 26.

getStringStoreProperty method

Gets a string value for a predefined or custom message store property.

Syntax

```
public virtual qa_int getStringStoreProperty(
     qa_const_string name,
    qa_string address,
    qa_int maxlen
)
```

Parameters

- **name** The predefined or custom property name.
- **address** The destination for the qa_string value.
- **maxlen** The maximum number of qa_chars of the value to copy, including the null terminator character.

Returns

The number of non-null qa_chars actually copied, or -1 if the operation failed.

Remarks

You can use this method to access predefined or user-defined client store properties.

For a list of predefined properties, see MessageStoreProperties.

For more information, see "Client message store properties" on page 26.

nextStorePropertyName method

Returns the message store property name for the given enumeration.

Syntax

```
public virtual qa_int nextStorePropertyName(
     qa_store_property_enum_handle h,
    qa_string buffer,
    qa_int bufferLen
)
```

Parameters

- **h** A handle returned by beginEnumStorePropertyNames.

- **buffer** The buffer into which to write the property name.
- **bufferLen** The length of the buffer to store the property name. This length must include space for the null terminator.

Returns

The length of the property name, or -1 if there are no more property names. property names

Remarks

If there are no more property names, returns -1.

See also

- "QAManagerBase.beginEnumStorePropertyNames method [QAnywhere C++]" on page 398

putMessage method

Puts a message into the queue for the given destination.

Syntax

```
public virtual qa_bool putMessage(
     qa_const_string address,
    QAMessage * msg
)
```

Parameters

- **address** The destination.
- **msg** The message.

Returns

True if and only if the operation succeeded.

putMessageTimeToLive method

Puts a message into the queue for the given destination and a given time-to-live in milliseconds.

Syntax

```
public virtual qa_bool putMessageTimeToLive(
     qa_const_string address,
    QAMessage * msg,
    qa_long ttl
)
```

Parameters

- **address** The destination.

- **msg** The message.
- **ttl** The time-to-live, in milliseconds.

Returns

True if and only if the operation succeeded.

setBooleanStoreProperty method

Sets a predefined or custom message store property to a boolean value.

Syntax

```
public virtual qa_bool setBooleanStoreProperty(
     qa_const_string name,
    qa_bool value
)
```

Parameters

- **name** The predefined or custom property name.
- **value** The qa_bool value of the property.

Returns

True if and only if the operation succeeded.

Remarks

You can use this method to set predefined or user-defined client. store properties.

For a list of predefined properties, see MessageStoreProperties.

For more information, see “Client message store properties” on page 26.

setByteStoreProperty method

Sets a predefined or custom message store property to a byte value.

Syntax

```
public virtual qa_bool setByteStoreProperty(
     qa_const_string name,
    qa_byte value
)
```

Parameters

- **name** The predefined or custom property name.
- **value** The qa_byte value of the property.

Returns

True if and only if the operation succeeded.

Remarks

You can use this method to set predefined or user-defined client store properties.

For a list of predefined properties, see MessageStoreProperties.

For more information, see "Client message store properties" on page 26.

setDoubleStoreProperty method

Sets a predefined or custom message store property to a double value.

Syntax

```
public virtual qa_bool setDoubleStoreProperty(
     qa_const_string name,
    qa_double value
)
```

Parameters

- **name** The predefined or custom property name.
- **value** The qa_double value of the property.

Returns

True if and only if the operation succeeded.

Remarks

You can use this method to set predefined or user-defined client store properties.

For a list of predefined properties, see MessageStoreProperties.

For more information, see "Client message store properties" on page 26.

setFloatStoreProperty method

Sets a predefined or custom message store property to a float value.

Syntax

```
public virtual qa_bool setFloatStoreProperty(
     qa_const_string name,
    qa_float value
)
```


Parameters

- **name** The predefined or custom property name.
- **value** The qa_float value of the property.

Returns

True if and only if the operation succeeded.

Remarks

You can use this method to set predefined or user-defined client store properties.

For a list of predefined properties, see MessageStoreProperties.

For more information, see "Client message store properties" on page 26.

setIntStoreProperty method

Sets a predefined or custom message store property to an int value.

Syntax

```
public virtual qa_bool setIntStoreProperty(
     qa_const_string name,
    qa_int value
)
```

Parameters

- **name** The predefined or custom property name.
- **value** The qa_int value of the property.

Returns

True if and only if the operation succeeded.

Remarks

You can use this method to set predefined or user-defined client store properties.

For a list of predefined properties, see MessageStoreProperties.

For more information, see "Client message store properties" on page 26.

setLongStoreProperty method

Sets a predefined or custom message store property to a long value.

Syntax

```
public virtual qa_bool setLongStoreProperty(
     qa_const_string name,
    qa_long value
)
```

Parameters

- **name** The predefined or custom property name.
- **value** The qa_long value of the property.

Returns

True if and only if the operation succeeded.

Remarks

You can use this method to set predefined or user-defined client store properties.

For a list of predefined properties, see MessageStoreProperties.

For more information, see "Client message store properties" on page 26.

setMessageListener method

Sets a message listener class to receive QAnywhere messages asynchronously.

Syntax

```
public virtual void setMessageListener(
     qa_const_string address,
    QAMessageListener * listener
)
```

Parameters

- **address** The destination address that the listener applies to.
- **listener** The message listener to associate with destination address.

Remarks

The listener is an instance of a class implementing QAMessageListener::onMessage, the only method defined in the QAMessageListener interface. QAMessageListener::onMessage accepts a single QAMessage parameter.

The setMessageListener address parameter specifies a local queue name used to receive the message. You can only have one listener assigned to a given queue.

If you want to listen for QAnywhere system messages, including push notifications and network status changes, specify "system" as the queue name. Use this method to receive message asynchronously.

For more information, see "Receiving messages asynchronously" on page 69 and "System queue" on page 58.

setMessageListenerBySelector method

Sets a message listener class to receive QAnywhere messages asynchronously, with a message selector.

Syntax

```
public virtual void setMessageListenerBySelector(
     qa_const_string address,
    qa_const_string selector,
    QAMessageListener * listener
)
```

Parameters

- **address** The destination address that the listener applies to.
- **selector** The selector to be used to filter the messages to be received.
- **listener** The message listener to associate with destination address.

Remarks

The listener is an instance of a class implementing QAMessageListener::onMessage, the only method defined in the QAMessageListener interface. QAMessageListener::onMessage accepts a single QAMessage parameter.

The setMessageListener address parameter specifies a local queue name used to receive the message. You can only have one listener assigned to a given queue. The selector parameter specifies a selector to be used to filter the messages to be received on the given address.

If you want to listen for QAnywhere system messages, including push notifications and network status changes, specify "system" as the queue name. Use this method to receive message asynchronously.

For more information, see "Receiving messages asynchronously" on page 69 and "System queue" on page 58.

setProperty method

Allows you to set QAnywhere manager configuration properties programmatically.

Syntax

```
public virtual qa_bool setProperty(
     qa_const_string name,
    qa_const_string value
)
```

Parameters

- **name** The predefined or custom QAnywhere Manager configuration property name.
- **value** The value of the QAnywhere Manager configuration property.

Returns

True if and only if the operation succeeded.

Remarks

You can use this method to override default QAnywhere manager configuration properties by specifying a property name and value.

For a list of QAnywhere manager configuration properties, see "QAnywhere manager configuration properties" on page 80.

You can also set QAnywhere manager configuration properties using a properties file and the QAManagerFactory::createQAManager method.

For more information, see "QAnywhere manager configuration property settings in a file" on page 82.

Note
You must set required properties before calling QAManager::open() or QATransactionalManager::open().

setShortStoreProperty method

Sets a predefined or custom message store property to a short value.

Syntax

```
public virtual qa_bool setShortStoreProperty(
     qa_const_string name,
    qa_short value
)
```

Parameters

- **name** The predefined or custom property name.
- **value** The qa_short value of the property.

Returns

True if and only if the operation succeeded.

Remarks

You can use this method to set predefined or user-defined client store properties.

For a list of predefined properties, see MessageStoreProperties.

For more information, see "Client message store properties" on page 26.

setStringStoreProperty method

Sets a predefined or custom message store property to a string value.

Syntax

```
public virtual qa_bool setStringStoreProperty(
     qa_const_string name,
    qa_const_string value
)
```

Parameters

- **name** The predefined or custom property name.
- **value** The qa_string value of the property.

Returns

True if and only if the operation succeeded.

Remarks

You can use this method to set predefined or user-defined client store properties.

For a list of predefined properties, see MessageStoreProperties.

For more information, see "Client message store properties" on page 26.

start method

Starts the QAManagerBase for receiving incoming messages in message listeners.

Syntax

```
public virtual qa_bool start()
```

Returns

True if and only if the operation succeeded.

Remarks

The QAManagerBase does not need to be started if there are no message listeners set, that is, if messages are received with the getMessage methods. It is not recommended to use the getMessage methods as well as message listeners for receiving messages. Use one or the other of the asynchronous (message listener) or synchronous (getMessage) models.

Any calls to start beyond the first without an intervening QAManagerBase::stop() call are ignored.

See also

- "QAManagerBase.stop method [QAnywhere C++]" on page 424

stop method

Stops the QAManagerBase's reception of incoming messages.

Syntax

```
public virtual qa_bool stop()
```

Returns

True if and only if the operation succeeded.

Remarks

The messages are not lost. They are not received until the manager is started again. Any calls to stop beyond the first without an intervening QAManagerBase::start() are ignored.

See also

- "QAManagerBase.start method [QAnywhere C++]" on page 423

triggerSendReceive method

Causes a synchronization with the QAnywhere message server, uploading any messages addressed to other clients, and downloading any messages addressed to the local client.

Syntax

```
public virtual qa_bool triggerSendReceive()
```

Returns

True if and only if the operation succeeded.

Remarks

A call to triggerSendReceive results in immediate message synchronization between a QAnywhere Agent and the central messaging server. A manual triggerSendReceive call results in immediate message transmission, independent of the QAnywhere Agent transmission policies. QAnywhere Agent transmission policies determine how message transmission occurs. For example, message transmission can occur automatically at regular intervals, when your client receives a push notification, or when you call the QAManagerBase::putMessage to send a message.

For more information, see "Determining when message transmission should occur on the client" on page 46.

See also

- "QAManagerBase.putMessage method [QAnywhere C++]" on page 416

QAManagerFactory class

This class acts as a factory class for creating QATransactionalManager and QAManager objects.

Syntax

```
public class QAManagerFactory
```

Members

All members of QAManagerFactory class, including all inherited members.

Name	Description
createQAManager method	Returns a new QAManager instance with the specified properties.
createQATransactionalManager method	Returns a new QATransactionalManager instance with the specified properties.
deleteQAManager method	Destroys a QAManager, freeing its resources.
deleteQATransactionalManager method	Destroys a QATransactionalManager instance, freeing its resources.
getLastError method	The error code associated with the last executed QAManagerFactory method.
getLastErrorMsg method	The error text associated with the last executed QAManagerFactory method.
getLastNativeError method	The native error code associated with the last executed QAManagerFactory method.

Remarks

You can only have one instance of QAManagerFactory.

See also

- "QAManager class [QAnywhere C++]" on page 388
- "QATransactionalManager class [QAnywhere C++]" on page 457

createQAManager method

Returns a new QAManager instance with the specified properties.

Syntax

```
public virtual QAManager * createQAManager(qa_const_string iniFile)
```

Parameters

- **iniFile** The path of the properties file.

Returns

The QAManager instance.

Remarks

If the properties file parameter is null, the QAManager is created using default properties. You can use the QAManager::setProperty() method to set QAnywhere Manager properties programmatically after you create the QAManager.

For a list of QAnywhere Manager configuration properties, see "QAnywhere manager configuration properties" on page 80.

See also

- "QAManager class [QAnywhere C++]" on page 388

createQATransactionalManager method

Returns a new QATransactionalManager instance with the specified properties.

Syntax

```
public virtual QATransactionalManager * createQATransactionalManager(
     qa_const_string iniFile
)
```

Parameters

- **iniFile** The path of the properties file.

Returns

The QATransactionalManager instance.

Remarks

If the properties file parameter is null, the QATransactionalManager is created using default properties. You can use the QATransactionalManager::setProperty() method to set QAnywhere Manager properties programmatically after you create the QATransactionalManager.

For a list of QAnywhere Manager configuration properties, see "QAnywhere manager configuration properties" on page 80.

See also

- "QATransactionalManager class [QAnywhere C++]" on page 457

deleteQAManager method

Destroys a QAManager, freeing its resources.

Syntax

```
public virtual void deleteQAManager(QAManager * mgr)
```

Parameters

- **mgr** The QAManager instance to destroy.

Remarks

It is not necessary to use this method, since all created QAManager's are destroyed when QAnywhereFactory_term() is called. It is provided as a convenience for when it is desirable to free resources in a timely manner.

For more information, see "Shutting down QAnywhere" on page 79.

deleteQATransactionalManager method

Destroys a QATransactionalManager instance, freeing its resources.

Syntax

```
public virtual void deleteQATransactionalManager(
     QATransactionalManager * mgr
)
```

Parameters

- **mgr** The QATransactionalManager instance to destroy.

Remarks

It is not necessary to use this method, since all created QATransactionalManager instances are destroyed when QAnywhereFactory_term() is called. It is provided as a convenience for when it is desirable to free resources in a timely manner.

For more information, see "Shutting down QAnywhere" on page 79

getLastError method

The error code associated with the last executed QAManagerFactory method.

Syntax

```
public virtual qa_int getLastError()
```

Returns

The error code.

Remarks

0 indicates no error.

For a list of values, see the QAError.

See also

- "QAManagerFactory.getLastNativeError method [QAnywhere C++]" on page 428
- "QAManagerFactory.getLastErrorMsg method [QAnywhere C++]" on page 428

getLastErrorMsg method

The error text associated with the last executed QAManagerFactory method.

Syntax

```
public virtual qa_string getLastErrorMsg()
```

Returns

The error message.

Remarks

This method returns null if QAManagerFactory::getLastError returns 0.

You can retrieve this property after catching a QAError.

See also

- "QAManagerFactory.getLastError method [QAnywhere C++]" on page 427
- "QAManagerFactory.getLastNativeError method [QAnywhere C++]" on page 428
- "QAError class [QAnywhere C++]" on page 380

getLastNativeError method

The native error code associated with the last executed QAManagerFactory method.

Syntax

```
public virtual an_sql_code getLastNativeError()
```

Returns

The native error code.

Remarks

0 indicates no error.

See also

- “QAManagerFactory.getLastError method [QAnywhere C++]” on page 427
- “QAManagerFactory.getLastErrorMsg method [QAnywhere C++]” on page 428

QAMessage class

QAMessage provides an interface to set message properties and header fields.

Syntax

```
public class QAMessage
```

Derived classes

- “QABinaryMessage class [QAnywhere C++]” on page 367
- “QATextMessage class [QAnywhere C++]” on page 452

Members

All members of QAMessage class, including all inherited members.

Name	Description
beginEnumPropertyNames method	Begins an enumeration of message property names.
castToBinaryMessage method	Casts this QAMessage to a QABinaryMessage.
castToTextMessage method	Casts this QAMessage to a QATextMessage.
clearProperties method	Clears a message's properties.
endEnumPropertyNames method	Frees the resources associated with a message property name enumeration.
getAddress method	Gets the destination address for the QAMessage instance.
getBooleanProperty method	Gets the value of the qa_bool property with the specified name.
getByteProperty method	Gets the value of the qa_byte property with the specified name.
getDoubleProperty method	Gets the value of the qa_double property with the specified name.
getExpiration method	Gets the message's expiration time.

Name	Description
getFloatProperty method	Gets the value of the qa_float property with the specified name.
getInReplyToID method	Gets the ID of the message that this message is in reply to.
getIntProperty method	Gets the value of the qa_int property with the specified name.
getLongProperty method	Gets the value of the qa_long property with the specified name.
getMessageID method	Gets the message ID.
getPriority method	Gets the message priority level.
getPropertyType method	Returns the type of a property with the given name.
getRedelivered method	Indicates whether the message has been previously received but not acknowledged.
getReplyToAddress method	Gets the address to which a reply to this message should be sent.
getShortProperty method	Gets the value of the qa_short property with the specified name.
getStringProperty method	Gets the value of the qa_string property with the specified name.
getTimestamp method	Gets the message timestamp.
getTimestampAsString method	Gets the message timestamp as a formatted string.
nextPropertyName method	Returns the message property name for the given enumeration, returning -1 if there are no more property names.
propertyExists method	Indicates whether a property value exists.
setAddress method	Sets the destination address for this message.
setBooleanProperty method	Sets the qa_bool property with the specified name to the specified value.
setByteProperty method	Sets a qa_byte property with the specified name to the specified value.
setDoubleProperty method	Sets the qa_double property with the specified name to the specified value.

Name	Description
setFloatProperty method	Sets the qa_float property with the specified name to the specified value.
setInReplyToID method	Sets the In-Reply-To ID for the message.
setIntProperty method	Sets the qa_int property with the specified name to the specified value.
setLongProperty method	Sets the qa_long property with the specified name to the specified value.
setMessageID method	Sets the message ID.
setPriority method	Sets the priority level for this message.
setRedelivered method	Sets an indication of whether this message was redelivered.
setReplyToAddress method	Sets the address to which a reply to this message should be sent.
setShortProperty method	Sets e qa_short property with the specified name to the specified value.
setStringProperty method	Sets a qa_string property with the specified name to the specified value.
setTimestamp method	Sets the message timestamp.
DEFAULT_PRIORITY variable	The default message priority.
DEFAULT_TIME_TO_LIVE variable	The default message time-to-live value.

Remarks

The derived classes QABinaryMessage and QATextMessage provide specialized methods to read and write to the message body. You can use QAMessage methods to set predefined or custom message properties.

For a list of predefined property names, see the MessageProperties.

For more information about setting message properties and header fields, see "QAnywhere messages" on page 13.

beginEnumPropertyNames method

Begins an enumeration of message property names.

Syntax

```
public virtual qa_property_enum_handle beginEnumPropertyNames()
```

Returns

A handle that is supplied to nextPropertyName.

Remarks

The handle returned by this method is supplied to nextPropertyName. This method and nextPropertyName can be used to enumerate the message property names at the time this method was called. Message properties cannot be set between beginEnumPropertyNames and endEnumPropertyNames.

castToBinaryMessage method

Casts this QAMessage to a QABinaryMessage.

Syntax

```
public virtual QABinaryMessage * castToBinaryMessage()
```

Returns

A pointer to the QABinaryMessage, or NULL if this message is not an instance of QABinaryMessage.

Remarks

You can also use the conversion operator to convert this QAMessage to a QABinaryMessage.

To convert a QAMessage to a QABinaryMessage using the conversion operator, do the following:

```
QAMessage *msg;
QABinaryMessage *bmsg;
...
bmsg = (QABinaryMessage *)(*msg);
```

castToTextMessage method

Casts this QAMessage to a QATextMessage.

Syntax

```
public virtual QATextMessage * castToTextMessage()
```

Returns

A pointer to the QATextMessage, or NULL if this message is not an instance of QATextMessage.

Remarks

You can also use the conversion operator to convert this QAMessage to a QATextMessage.

For example, to convert a QAMessage to a QATextMessage using the conversion operator, do the following:

```
QAMessage *msg;
QATextMessage *bmsg;
...
bmsg = (QATextMessage *)(*msg);
```

clearProperties method

Clears a message's properties.

Syntax

```
public virtual void clearProperties()
```

Remarks

Note
The message's header fields and body are not cleared.

endEnumPropertyNames method

Frees the resources associated with a message property name enumeration.

Syntax

```
public virtual void endEnumPropertyNames(qa_property_enum_handle h)
```

Parameters

- **h** A handle returned by beginEnumPropertyNames.

getAddress method

Gets the destination address for the QAMessage instance.

Syntax

```
public virtual qa_const_string getAddress()
```

Returns

The destination address.

Remarks

When a message is sent, this field is ignored. After completion of the send method, the field holds the destination address specified in QAManagerBase::putMessage().

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

getBooleanProperty method

Gets the value of the qa_bool property with the specified name.

Syntax

```
public virtual qa_bool getBooleanProperty(
     qa_const_string name,
    qa_bool * value
)
```

Parameters

- **name** The name of the property to get.
- **value** The destination for the qa_bool value.

Returns

True if and only if the operation succeeded.

Remarks

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

See also

- "MessageProperties class [QAnywhere C++]" on page 356

getByteProperty method

Gets the value of the qa_byte property with the specified name.

Syntax

```
public virtual qa_bool getByteProperty(
     qa_const_string name,
    qa_byte * value
)
```

Parameters

- **name** The name of the property to get.
- **value** The destination for the qa_byte value.

Returns

True if and only if the operation succeeded.

Remarks

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

See also

- "MessageProperties class [QAnywhere C++]" on page 356

getDoubleProperty method

Gets the value of the qa_double property with the specified name.

Syntax

```
public virtual qa_bool getDoubleProperty(
     qa_const_string name,
    qa_double * value
)
```

Parameters

- **name** The name of the property to get.
- **value** The destination for the qa_double value.

Returns

True if and only if the operation succeeded.

Remarks

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

See also

- "MessageProperties class [QAnywhere C++]" on page 356

getExpiration method

Gets the message's expiration time.

Syntax

```
public virtual qa_long getExpiration()
```

Returns

The expiration time.

Remarks

When a message is sent, the Expiration header field is left unassigned. After the send method completes, the Expiration header holds the expiration time of the message.

This property is read-only because the expiration time of a message is set by adding the time-to-live argument of QAManagerBase::putMessageTimeToLive to the current time.

The expiration time is in units that are natural for the platform. For Windows/PocketPC platforms, expiration is a SYSTEMTIME, converted to a FILETIME, which is copied to an qa_long value.

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

See also

- "QAMessage.getTimestamp method [QAnywhere C++]" on page 442

getFloatProperty method

Gets the value of the qa_float property with the specified name.

Syntax

```
public virtual qa_bool getFloatProperty(
     qa_const_string name,
    qa_float * value
)
```

Parameters

- **name** The name of the property to get.
- **value** The destination for the qa_float value.

Returns

True if and only if the operation succeeded.

Remarks

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

See also

- "MessageProperties class [QAnywhere C++]" on page 356

getInReplyToID method

Gets the ID of the message that this message is in reply to.

Syntax

```
public virtual qa_const_string getInReplyToID()
```

Returns

The In-Reply-To ID.

Remarks

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

getIntProperty method

Gets the value of the qa_int property with the specified name.

Syntax

```
public virtual qa_bool getIntProperty(
     qa_const_string name,
    qa_int * value
)
```

Parameters

- **name** The name of the property to get.
- **value** The destination for the qa_int value.

Returns

True if and only if the operation succeeded.

Remarks

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

See also

- "MessageProperties class [QAnywhere C++]" on page 356

getLongProperty method

Gets the value of the qa_long property with the specified name.

Syntax

```
public virtual qa_bool getLongProperty(
     qa_const_string name,
    qa_long * value
)
```

Parameters

- **name** The name of the property to get.
- **value** The destination for the qa_long value.

Returns

True if and only if the operation succeeded.

Remarks

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

See also

- "MessageProperties class [QAnywhere C++]" on page 356

getMessageID method

Gets the message ID.

Syntax

```
public virtual qa_const_string getMessageID()
```

Returns

The message ID.

Remarks

The MessageID header field contains a value that uniquely identifies each message sent by the QAnywhere client.

When a message is sent using QAManagerBase::putMessage method, the MessageID header is null and can be ignored. When the send method returns, it contains an assigned value.

A MessageID is a qa_string value that should function as a unique key for identifying messages in a historical repository.

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

getPriority method

Gets the message priority level.

Syntax

```
public virtual qa_int getPriority()
```


Returns

The message priority.

Remarks

The QAnywhere client API defines ten levels of priority value, with 0 as the lowest priority and 9 as the highest. Clients should consider priorities 0-4 as gradations of normal priority and priorities 5-9 as gradations of expedited priority.

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

getPropertyType method

Returns the type of a property with the given name.

Syntax

```
public virtual qa_short getPropertyType(qa_const_string name)
```

Parameters

- **name** The name of the property.

Returns

The type of the property.

Remarks

One of PROPERTY_TYPE_BOOLEAN, PROPERTY_TYPE_BYTE, PROPERTY_TYPE_SHORT, PROPERTY_TYPE_INT, PROPERTY_TYPE_LONG, PROPERTY_TYPE_FLOAT, PROPERTY_TYPE_DOUBLE, PROPERTY_TYPE_STRING, PROPERTY_TYPE_UNKNOWN.

getRedelivered method

Indicates whether the message has been previously received but not acknowledged.

Syntax

```
public virtual qa_bool getRedelivered()
```

Returns

True if and only if the message was redelivered.

Remarks

The Redelivered header is set by a receiving QAManager when it detects that a message being received was received before.

For example, an application receives a message using a QAManager opened with AcknowledgementMode::EXPLICIT_ACKNOWLEDGEMENT, and shuts down without acknowledging the message. When the application starts again and receives the same message the Redelivered header is true.

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

getReplyToAddress method

Gets the address to which a reply to this message should be sent.

Syntax

```
public virtual qa_const_string getReplyToAddress()
```

Returns

The reply-to address.

Remarks

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

getShortProperty method

Gets the value of the qa_short property with the specified name.

Syntax

```
public virtual qa_bool getShortProperty(
     qa_const_string name,
    qa_short * value
)
```

Parameters

- **name** The name of the property to get.
- **value** The destination for the qa_short value.

Returns

True if and only if the operation succeeded.

Remarks

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

See also

- "MessageProperties class [QAnywhere C++]" on page 356

getStringProperty method

Gets the value of the qa_string property with the specified name.

Overload list

Name	Description
getStringProperty(qa_const_string, qa_int, qa_string, qa_int) method	Gets the value of the qa_string property (starting at offset) with the specified name.
getStringProperty(qa_const_string, qa_string, qa_int) method	Gets the value of the qa_string property with the specified name.

getStringProperty(qa_const_string, qa_int, qa_string, qa_int) method

Gets the value of the qa_string property (starting at offset) with the specified name.

Syntax

```
public virtual qa_int getStringProperty(
     qa_const_string name,
    qa_int offset,
    qa_string dest,
    qa_int maxlen
)
```

Parameters

- **name** The name of the property to get.
- **offset** The starting offset into the property value from which to copy.
- **dest** The destination for the qa_string value.
- **maxlen** The maximum number of qa_chars of the value to copy. This value includes the null terminator qa_char.

Returns

The number of non-null qa_chars actually copied, or -1 if the operation failed.

Remarks

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

See also

- "MessageProperties class [QAnywhere C++]" on page 356

getStringProperty(qa_const_string, qa_string, qa_int) method

Gets the value of the qa_string property with the specified name.

Syntax

```
public virtual qa_int getStringProperty(
     qa_const_string name,
    qa_string dest,
    qa_int maxlen
)
```

Parameters

- **name** The name of the property to get.
- **dest** The destination for the qa_string value.
- **maxlen** The maximum number of qa_chars of the value to copy. This value includes the null terminator qa_char.

Returns

The number of non-null qa_chars actually copied, or -1 if the operation failed.

Remarks

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

See also

- "MessageProperties class [QAnywhere C++]" on page 356

getTimestamp method

Gets the message timestamp.

Syntax

```
public virtual qa_long getTimestamp()
```

Returns

The message timestamp.

Remarks

This Timestamp header field contains the time a message was created. It is a coordinated universal time (UTC).

It is not the time the message was actually transmitted, because the actual send may occur later due to transactions or other client-side queuing of messages. It is in units that are natural for the platform. For Windows/PocketPC platforms, the timestamp is a SYSTEMTIME, converted to a FILETIME, which is copied to a qa_long value.

To convert a timestamp ts to SYSTEMTIME for displaying to a user, run the following code:

```
SYSTEMTIME stime;
FILETIME   ftime;
ULARGE_INTEGER time;
time.QuadPart = ts;
memcpy(&ftime, &time, sizeof(FILETIME));
FileTimeToSystemTime(&ftime, &stime);
```

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

getTimestampAsString method

Gets the message timestamp as a formatted string.

Syntax

```
public virtual qa_int getTimestampAsString(
     qa_string buffer,
    qa_int bufferLen
)
```

Parameters

- **buffer** The buffer for the formatted timestamp.
- **bufferLen** The size of the buffer.

Returns

The number of non-null qa_chars written to the buffer.

Remarks

The format is: "dow, MMM dd, yyyy hh:mm:ss.nnn GMT".

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

nextPropertyName method

Returns the message property name for the given enumeration, returning -1 if there are no more property names.

Syntax

```
public virtual qa_int nextPropertyName(
     qa_property_enum_handle h,
    qa_string buffer,
    qa_int bufferLen
)
```

Parameters

- **h** A handle returned by beginEnumPropertyNames.
- **buffer** The buffer into which to write the property name.
- **bufferLen** The length of the buffer to store the property name. This length must include space for the null terminator

Returns

The length of the property name, or -1 if there are no more property names.

propertyExists method

Indicates whether a property value exists.

Syntax

```
public virtual qa_bool propertyExists(qa_const_string name)
```

Parameters

- **name** The name of the property.

Returns

True if and only if the property exists.

setAddress method

Sets the destination address for this message.

Syntax

```
public virtual void setAddress(qa_const_string destination)
```


Parameters

- **destination** The destination address.

Remarks

This method can be used to change the value for a message that has been received.

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

setBooleanProperty method

Sets the qa_bool property with the specified name to the specified value.

Syntax

```
public virtual void setBooleanProperty(
     qa_const_string name,
    qa_bool value
)
```

Parameters

- **name** the name of the property to set.
- **value** the qa_bool value of the property.

Remarks

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

See also

- "MessageProperties class [QAnywhere C++]" on page 356

setByteProperty method

Sets a qa_byte property with the specified name to the specified value.

Syntax

```
public virtual void setByteProperty(qa_const_string name, qa_byte value)
```

Parameters

- **name** The name of the property to set.
- **value** The qa_byte value of the property.

Remarks

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

See also

- "MessageProperties class [QAnywhere C++]" on page 356

setDoubleProperty method

Sets the qa_double property with the specified name to the specified value.

Syntax

```
public virtual void setDoubleProperty(
     qa_const_string name,
    qa_double value
)
```

Parameters

- **name** The name of the property to set.
- **value** The qa_double value of the property.

Remarks

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

See also

- "MessageProperties class [QAnywhere C++]" on page 356

setFloatProperty method

Sets the qa_float property with the specified name to the specified value.

Syntax

```
public virtual void setFloatProperty(
     qa_const_string name,
    qa_float value
)
```

Parameters

- **name** The name of the property to set.
- **value** The qa_float value of the property.

Remarks

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

See also

- "MessageProperties class [QAnywhere C++]" on page 356

setInReplyToID method

Sets the In-Reply-To ID for the message.

Syntax

```
public virtual void setInReplyToID(qa_const_string id)
```

Parameters

- **id** The In-Reply-To ID.

Remarks

A client can use the InReplyToID header field to link one message with another. A typical use is to link a response message with its request message.

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

setIntProperty method

Sets the qa_int property with the specified name to the specified value.

Syntax

```
public virtual void setIntProperty(qa_const_string name, qa_int value)
```

Parameters

- **name** The name of the property to set.
- **value** The qa_int value of the property.

Remarks

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

See also

- "MessageProperties class [QAnywhere C++]" on page 356

setLongProperty method

Sets the qa_long property with the specified name to the specified value.

Syntax

```
public virtual void setLongProperty(qa_const_string name, qa_long value)
```

Parameters

- **name** The name of the property to set.
- **value** The qa_long value of the property.

Remarks

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

See also

- "MessageProperties class [QAnywhere C++]" on page 356

setMessageID method

Sets the message ID.

Syntax

```
public virtual void setMessageID(qa_const_string id)
```

Parameters

- **id** The message ID.

Remarks

This method can be used to change the value for a message that has been received.

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

setPriority method

Sets the priority level for this message.

Syntax

```
public virtual void setPriority(qa_int priority)
```


Parameters

- **priority** The message priority.

Remarks

This method can be used to change the value for a message that has been received.

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

setRedelivered method

Sets an indication of whether this message was redelivered.

Syntax

```
public virtual void setRedelivered(qa_bool redelivered)
```

Parameters

- **redelivered** The redelivered indication.

Remarks

This method can be used to change the value for a message that has been received.

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

setReplyToAddress method

Sets the address to which a reply to this message should be sent.

Syntax

```
public virtual void setReplyToAddress(qa_const_string replyTo)
```

Parameters

- **replyTo** The reply-to address.

Remarks

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

setShortProperty method

Sets e qa_short property with the specified name to the specified value.

Syntax

```
public virtual void setShortProperty(
     qa_const_string name,
    qa_short value
)
```

Parameters

- **name** The name of the property to set.
- **value** The qa_short value of the property.

Remarks

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

See also

- "MessageProperties class [QAnywhere C++]" on page 356

setStringProperty method

Sets a qa_string property with the specified name to the specified value.

Syntax

```
public virtual void setStringProperty(
     qa_const_string name,
    qa_const_string value
)
```

Parameters

- **name** The name of the property to set.
- **value** The qa_string value of the property.

Remarks

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

See also

- "MessageProperties class [QAnywhere C++]" on page 356

setTimestamp method

Sets the message timestamp.

Syntax

```
public virtual void setTimestamp(qa_long timestamp)
```

Parameters

- **timestamp** The message timestamp, a coordinated universal time (UTC).

Remarks

This method can be used to change the value for a message that has been received.

For more information about getting and setting message headers and properties, see "QAnywhere messages" on page 13.

See also

- "QAMessage.getTimestamp method [QAnywhere C++]" on page 442

DEFAULT_PRIORITY variable

The default message priority.

Syntax

```
public static const qa_int DEFAULT_PRIORITY;
```

Remarks

This value is 4. This is normal priority as values 0-4 are gradations of normal priority and values 5-9 are gradations of expedited priority.

DEFAULT_TIME_TO_LIVE variable

The default message time-to-live value.

Syntax

```
public static const qa_long DEFAULT_TIME_TO_LIVE;
```

Remarks

This value is 0, which indicates that the message does not expire.

QAMessageListener class

A QAMessageListener object is used to receive asynchronously delivered messages.

Syntax

```
public class QAMessageListener
```

Members

All members of QAMessageListener class, including all inherited members.

Name	Description
QAMessageListener deconstructor	Virtual destructor.
onMessage method	Passes a message to the listener.

QAMessageListener deconstructor

Virtual destructor.

Syntax

```
public virtual ~QAMessageListener()
```

onMessage method

Passes a message to the listener.

Syntax

```
public virtual void onMessage(QAMessage * message)
```

Parameters

- **message** The message passed to the listener.

QATextMessage class

QATextMessage inherits from the QAMessage class and adds a text message body.

Syntax

```
public class QATextMessage : QAMessage
```

Base classes

- "QAMessage class [QAnywhere C++]" on page 429

Members

All members of QATextMessage class, including all inherited members.

Name	Description
beginEnumPropertyNames method	Begins an enumeration of message property names.
castToBinaryMessage method	Casts this QAMessage to a QABinaryMessage.
castToTextMessage method	Casts this QAMessage to a QATextMessage.
clearProperties method	Clears a message's properties.
endEnumPropertyNames method	Frees the resources associated with a message property name enumeration.
getAddress method	Gets the destination address for the QAMessage instance.
getBooleanProperty method	Gets the value of the qa_bool property with the specified name.
getByteProperty method	Gets the value of the qa_byte property with the specified name.
getDoubleProperty method	Gets the value of the qa_double property with the specified name.
getExpiration method	Gets the message's expiration time.
getFloatProperty method	Gets the value of the qa_float property with the specified name.
getInReplyToID method	Gets the ID of the message that this message is in reply to.
getIntProperty method	Gets the value of the qa_int property with the specified name.
getLongProperty method	Gets the value of the qa_long property with the specified name.
getMessageID method	Gets the message ID.
getPriority method	Gets the message priority level.
getPropertyType method	Returns the type of a property with the given name.
getRedelivered method	Indicates whether the message has been previously received but not acknowledged.
getReplyToAddress method	Gets the address to which a reply to this message should be sent.

Name	Description
getShortProperty method	Gets the value of the qa_short property with the specified name.
getStringProperty method	Gets the value of the qa_string property with the specified name.
getText method	Gets the string containing this message's data.
getTextLength method	Returns the text length.
getTimestamp method	Gets the message timestamp.
getTimestampAsString method	Gets the message timestamp as a formatted string.
nextPropertyName method	Returns the message property name for the given enumeration, returning -1 if there are no more property names.
propertyExists method	Indicates whether a property value exists.
readText method	Reads the requested length of text from the current text position into a buffer.
reset method	Repositions the current text position to the beginning.
setAddress method	Sets the destination address for this message.
setBooleanProperty method	Sets the qa_bool property with the specified name to the specified value.
setByteProperty method	Sets a qa_byte property with the specified name to the specified value.
setDoubleProperty method	Sets the qa_double property with the specified name to the specified value.
setFloatProperty method	Sets the qa_float property with the specified name to the specified value.
setInReplyToID method	Sets the In-Reply-To ID for the message.
setIntProperty method	Sets the qa_int property with the specified name to the specified value.
setLongProperty method	Sets the qa_long property with the specified name to the specified value.
setMessageID method	Sets the message ID.

Name	Description
setPriority method	Sets the priority level for this message.
setRedelivered method	Sets an indication of whether this message was redelivered.
setReplyToAddress method	Sets the address to which a reply to this message should be sent.
setShortProperty method	Sets e qa_short property with the specified name to the specified value.
setStringProperty method	Sets a qa_string property with the specified name to the specified value.
setText method	Sets the string containing this message's data.
setTimestamp method	Sets the message timestamp.
writeText method	Concatenates text to the current text.
DEFAULT_PRIORITY variable	The default message priority.
DEFAULT_TIME_TO_LIVE variable	The default message time-to-live value.

Remarks

QATextMessage provides methods to read from and write to the text message body.

When the message is first created, the body of the message is in write-only mode. After a message has been sent, the client that sent it can retain and modify it without affecting the message that has been sent. The same message object can be sent multiple times.

When a message is received, the provider has called QATextMessage::reset so that the message body is in read-only mode and reading of values starts from the beginning of the message body. If a client attempts to write a message in read-only mode, a COMMON_MSG_NOT_WRITEABLE_ERROR is set.

See also

- "QABinaryMessage class [QAnywhere C++]" on page 367

getText method

Gets the string containing this message's data.

Syntax

```
public virtual qa_string getText()
```

Returns

A string containing the message's data.

Remarks

The default value is null.

If the message exceeds the maximum size specified by the QAManager::MAX_IN_MEMORY_MESSAGE_SIZE property, this function returns null. In this case, use the QATextMessage::readText method to read the text.

For more information about QAManager properties, see "QAnywhere manager configuration properties" on page 80.

getTextLength method

Returns the text length.

Syntax

```
public virtual qa_long getTextLength()
```

Returns

The text length.

Remarks

> **Note**
> If the text length is non-zero and getText() returns qa_null then the text does not fit in memory, and must be read in pieces using the readText.

readText method

Reads the requested length of text from the current text position into a buffer.

Syntax

```
public virtual qa_int readText(qa_string string, qa_int length)
```

Parameters

- **string** The destination for the text.
- **length** The maximum number of qa_chars to read into the destination. buffer, including the null termination character.

Returns

The actual number of non-null qa_chars read, or -1 if the entire text stream has been read.

reset method

Repositions the current text position to the beginning.

Syntax

```
public virtual void reset()
```

setText method

Sets the string containing this message's data.

Syntax

```
public virtual void setText(qa_const_string string)
```

Parameters

- **string** A string containing the message data to set.

writeText method

Concatenates text to the current text.

Syntax

```
public virtual void writeText(
     qa_const_string string,
    qa_int offset,
    qa_int length
)
```

Parameters

- **string** The source text to concatenate.
- **offset** The offset into the source text at which to start reading.
- **length** The number of qa_chars of the source text to read.

QATransactionalManager class

This class is the manager for transactional messaging.

Syntax

```
public class QATransactionalManager : QAManagerBase
```

Base classes

- "QAManagerBase class [QAnywhere C++]" on page 394

Members

All members of QATransactionalManager class, including all inherited members.

Name	Description
beginEnumStoreProperty-Names method	Begins an enumeration of message store property names.
browseClose method	Frees the resources associated with a browse operation.
browseMessages method	Begins a browse of messages queued in the message store.
browseMessagesByID meth-od	Begins a browse of the message that is queued in the message store, with the given message ID.
browseMessagesByQueue method	Begins a browse of messages queued in the message store for the given queue.
browseMessagesBySelector method	Begins a browse of messages queued in the message store that satisfy the given selector.
browseNextMessage method	Returns the next message for the given browse operation, returning null if there are no more messages.
cancelMessage method	Cancels the message with the given message ID.
close method	Closes the connection to the QAnywhere message system and releases any resources used by the QAManagerBase.
commit method	Commits the current transaction and begins a new transaction.
createBinaryMessage meth-od	Creates a QABinaryMessage instance.
createTextMessage method	Creates a QATextMessage instance.
deleteMessage method	Deletes a QAMessage object.
endEnumStoreProperty-Names method	Frees the resources associated with a message store property name enu-meration.
getAllQueueDepth method	Returns the total depth of all queues, based on a given filter.
getBooleanStoreProperty method	Gets a boolean value for a predefined or custom message store proper-ty.

Name	Description
getByteStoreProperty method	Gets a byte value for a predefined or custom message store property.
getDoubleStoreProperty method	Gets a double value for a predefined or custom message store property.
getFloatStoreProperty method	Gets a float value for a predefined or custom message store property.
getIntStoreProperty method	Gets an int value for a predefined or custom message store property.
getLastError method	The error code associated with the last executed QAManagerBase method.
getLastErrorMsg method	The error text associated with the last executed QAManagerBase method.
getLastNativeError method	The native error code associated with the last executed QAManagerBase method.
getLongStoreProperty method	Gets a long value for a predefined or custom message store property.
getMessage method	Returns the next available QAMessage sent to the specified address.
getMessageBySelector method	Returns the next available QAMessage sent to the specified address that satisfies the given selector.
getMessageBySelectorNoWait method	Returns the next available QAMessage sent to the given address that satisfies the given selector.
getMessageBySelectorTimeout method	Returns the next available QAMessage sent to the given address that satisfies the given selector.
getMessageNoWait method	Returns the next available QAMessage sent to the given address.
getMessageTimeout method	Returns the next available QAMessage sent to the given address.
getMode method	Returns the QAManager acknowledgement mode for received messages.
getQueueDepth method	Returns the depth of a queue, based on a given filter.
getShortStoreProperty method	Gets a short value for a predefined or custom message store property.

Name	Description
getStringStoreProperty method	Gets a string value for a predefined or custom message store property.
nextStorePropertyName method	Returns the message store property name for the given enumeration.
open method	Opens a QATransactionalManager instance.
putMessage method	Puts a message into the queue for the given destination.
putMessageTimeToLive method	Puts a message into the queue for the given destination and a given time-to-live in milliseconds.
rollback method	Rolls back the current transaction and begins a new transaction.
setBooleanStoreProperty method	Sets a predefined or custom message store property to a boolean value.
setByteStoreProperty method	Sets a predefined or custom message store property to a byte value.
setDoubleStoreProperty method	Sets a predefined or custom message store property to a double value.
setFloatStoreProperty method	Sets a predefined or custom message store property to a float value.
setIntStoreProperty method	Sets a predefined or custom message store property to an int value.
setLongStoreProperty method	Sets a predefined or custom message store property to a long value.
setMessageListener method	Sets a message listener class to receive QAnywhere messages asynchronously.
setMessageListenerBySelector method	Sets a message listener class to receive QAnywhere messages asynchronously, with a message selector.
setProperty method	Allows you to set QAnywhere manager configuration properties programmatically.
setShortStoreProperty method	Sets a predefined or custom message store property to a short value.
setStringStoreProperty method	Sets a predefined or custom message store property to a string value.

Name	Description
start method	Starts the QAManagerBase for receiving incoming messages in message listeners.
stop method	Stops the QAManagerBase's reception of incoming messages.
triggerSendReceive method	Causes a synchronization with the QAnywhere message server, uploading any messages addressed to other clients, and downloading any messages addressed to the local client.

Remarks

The QATransactionalManager class derives from QAManagerBase and manages transactional QAnywhere messaging operations.

For a detailed description of derived behavior, see QAManagerBase.

The QATransactionalManager can only be used for transactional acknowledgement. Use the QATransactionalManager::commit() method to commit all QAManagerBase::putMessage() and QAManagerBase::getMessage() invocations.

For more information, see "Transactional messaging implementation" on page 63

See also

- "QATransactionalManager class [QAnywhere C++]" on page 457

commit method

Commits the current transaction and begins a new transaction.

Syntax

```
public virtual qa_bool commit()
```

Returns

True if and only if the commit operation was successful.

Remarks

This method commits all QAManagerBase::putMessage() and QAManagerBase::getMessage() invocations.

> **Note**
> The first transaction begins with the call to QATransactionalManager::open().

See also

- "QATransactionalManager class [QAnywhere C++]" on page 457

open method

Opens a QATransactionalManager instance.

Syntax

```
public virtual qa_bool open()
```

Returns

True if and only if the operation was successful.

Remarks

The open method must be the first method called after creating a manager.

See also

- "QATransactionalManager class [QAnywhere C++]" on page 457

rollback method

Rolls back the current transaction and begins a new transaction.

Syntax

```
public virtual qa_bool rollback()
```

Returns

True if and only if the open operation was successful.

Remarks

This method rolls back all uncommitted QAManagerBase::putMessage() and QAManagerBase::getMessage() invocations.

See also

- "QATransactionalManager class [QAnywhere C++]" on page 457

QueueDepthFilter class

QueueDepthFilter values for queue depth methods of QAManagerBase.

Syntax

```
public class QueueDepthFilter
```

Members

All members of QueueDepthFilter class, including all inherited members.

Name	Description
ALL variable	Count both incoming and outgoing messages.
INCOMING variable	Count only incoming messages.
LOCAL variable	Count only local messages.
OUTGOING variable	Count only outgoing messages.

ALL variable

Count both incoming and outgoing messages.

Syntax

```
public static const qa_short ALL;
```

Remarks

System messages and expired messages are not included in any queue depth counts.

INCOMING variable

Count only incoming messages.

Syntax

```
public static const qa_short INCOMING;
```

Remarks

An incoming message is defined as a message whose originator is different than the agent ID of the message store.

LOCAL variable

Count only local messages.

Syntax

```
public static const qa_short LOCAL;
```

Remarks

A local message is defined as a message whose originator and target are the agent ID of the message store.

OUTGOING variable

Count only outgoing messages.

Syntax

```
public static const qa_short OUTGOING;
```

Remarks

An outgoing message is defined as a message whose originator is the agent ID of the message store, and whose destination is not the agent ID of the message store.

StatusCodes class

This interface defines a set of codes for the status of a message.

Syntax

```
public class StatusCodes
```

Members

All members of StatusCodes class, including all inherited members.

Name	Description
CANCELED variable	The message has been canceled.
EXPIRED variable	The message has expired, that is the message was not received before its expiration time passed.
FINAL variable	This constant is used to determine if a message has achieved a final state.
LOCAL variable	The message is addressed to the local message store and will not be transmitted to the server.
PENDING variable	The message has been sent but not received and acknowledged.
RECEIVED variable	The message has been received and acknowledged by the receiver.
RECEIVING variable	The message is in the process of being received, or it was received but not acknowledged.
TRANSMITTED variable	The message has been transmitted to the server.
TRANSMITTING variable	The message is in the process of being transmitted to the server.

Name	Description
UNRECEIVABLE variable	The message has been marked as unreceivable.
UNTRANSMITTED variable	The message has not been transmitted to the server.

CANCELED variable

The message has been canceled.

Syntax

```
public static const qa_int CANCELED;
```

Remarks

This code has value 40. This code applies to MessageProperties::STATUS.

EXPIRED variable

The message has expired, that is the message was not received before its expiration time passed.

Syntax

```
public static const qa_int EXPIRED;
```

Remarks

This code has value 30. This code applies to MessageProperties::STATUS.

FINAL variable

This constant is used to determine if a message has achieved a final state.

Syntax

```
public static const qa_int FINAL;
```

Remarks

A message has achieved a final state if and only if its status is greater than this constant.

This code has value 20. This code applies to MessageProperties::STATUS.

LOCAL variable

The message is addressed to the local message store and will not be transmitted to the server.

Syntax

```
public static const qa_int LOCAL;
```

Remarks

This code has value 2. This code applies to MessageProperties::TRANSMISSION_STATUS.

PENDING variable

The message has been sent but not received and acknowledged.

Syntax

```
public static const qa_int PENDING;
```

Remarks

This code has value 1. This code applies to MessageProperties::STATUS.

RECEIVED variable

The message has been received and acknowledged by the receiver.

Syntax

```
public static const qa_int RECEIVED;
```

Remarks

This code has value 60. This code applies to MessageProperties::STATUS.

RECEIVING variable

The message is in the process of being received, or it was received but not acknowledged.

Syntax

```
public static const qa_int RECEIVING;
```

Remarks

This code has value 10. This code applies to MessageProperties::STATUS.

TRANSMITTED variable

The message has been transmitted to the server.

Syntax

```
public static const qa_int TRANSMITTED;
```

Remarks

This code has value 1. This code applies to MessageProperties::TRANSMISSION_STATUS.

TRANSMITTING variable

The message is in the process of being transmitted to the server.

Syntax

```
public static const qa_int TRANSMITTING;
```

Remarks

This code has value 3. This code applies to MessageProperties::TRANSMISSION_STATUS.

UNRECEIVABLE variable

The message has been marked as unreceivable.

Syntax

```
public static const qa_int UNRECEIVABLE;
```

Remarks

The message is either malformed, or there were too many failed attempts to deliver it.

This code has value 50. This code applies to MessageProperties::STATUS.

UNTRANSMITTED variable

The message has not been transmitted to the server.

Syntax

```
public static const qa_int UNTRANSMITTED;
```

Remarks

This code has value 0. This code applies to MessageProperties::TRANSMISSION_STATUS.

QAnywhere Java API reference for clients

Package (for regular clients)

```
ianywhere.qanywhere.client
```

Package (for standalone clients)

```
ianywhere.qanywhere.standaloneclient
```

AcknowledgementMode interface

Indicates how messages should be acknowledged by QAnywhere client applications.

Syntax

```
public interface AcknowledgementMode
```

Members

All members of AcknowledgementMode interface, including all inherited members.

Name	Description
EXPLICIT_ACKNOWLEDGEMENT variable	Indicates that received messages are acknowledged using one of the QAManager acknowledge methods.
IMPLICIT_ACKNOWLEDGEMENT variable	Indicates that all messages are acknowledged as soon as they are received by a client application.
TRANSACTIONAL variable	This mode indicates that messages are only acknowledged as part of the on going transaction.

Remarks

The implicit and explicit acknowledgement modes are assigned to a QAManager instance using the QAManager.open(short) method.

With implicit acknowledgement, messages are acknowledged as soon as they are received by a client application. With explicit acknowledgement, you must call one of the QAManager acknowledgement methods. The server propagates all status changes from client to client.

See also

- "QAManager interface [QAnywhere Java]" on page 507
- "QATransactionalManager interface [QAnywhere Java]" on page 575
- "QAManagerBase interface [QAnywhere Java]" on page 514

EXPLICIT_ACKNOWLEDGEMENT variable

Indicates that received messages are acknowledged using one of the QAManager acknowledge methods.

Syntax

```
final short AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT
```

See also

- "QAManager interface [QAnywhere Java]" on page 507

IMPLICIT_ACKNOWLEDGEMENT variable

Indicates that all messages are acknowledged as soon as they are received by a client application.

Syntax

```
final short AcknowledgementMode.IMPLICIT_ACKNOWLEDGEMENT
```

Remarks

If you receive messages synchronously, messages are acknowledged as soon as the QAManagerBase.getMessage(String) method returns. If you receive messages asynchronously, the message is acknowledged as soon as the event handling method returns.

See also

- "QAManagerBase.getMessage method [QAnywhere Java]" on page 525

TRANSACTIONAL variable

This mode indicates that messages are only acknowledged as part of the on going transaction.

Syntax

```
final short AcknowledgementMode.TRANSACTIONAL
```

Remarks

This mode is automatically assigned to QATransactionalManager instances.

See also

- "QATransactionalManager interface [QAnywhere Java]" on page 575

MessageProperties interface

Provides fields storing standard message property names.

Syntax

```
public interface MessageProperties
```

Members

All members of MessageProperties interface, including all inherited members.

Name	Description
ADAPTER variable	For "system" queue messages, the network adapter that is being used to connect to the QAnywhere server.
ADAPTERS variable	This property name refers to a delimited list of network adapters that can be used to connect to the QAnywhere server.
DELIVERY_COUNT variable	This property name refers to the number of attempts that have been made so far to deliver the message.
IP variable	For "system" queue messages, the IP address of the network adapter that is being used to connect to the QAnywhere server.
MAC variable	For "system" queue messages, the MAC address of the network adapter that is being used to connect to the QAnywhere server.
MSG_TYPE variable	This property name refers to MessageType enumeration values associated with a QAnywhere message.
NETWORK_STATUS variable	This property name refers to the state of the network connection.
ORIGINATOR variable	This property name refers to the message store ID of the originator of the message.
RAS variable	For "system" queue messages, the RAS entry name that is being used to connect to the QAnywhere server.
RASNAMES variable	For "system" queue messages, a delimited list of RAS entry names that can be used to connect to the QAnywhere server.
STATUS variable	This property name refers to the current status of the message.
STATUS_TIME variable	This property name refers to the time at which the message assumed its current status.
TRANSMISSION_STATUS variable	This property name refers to the current transmission status of the message.

Remarks

The MessageProperties class provides standard message property names. You can pass MessageProperties fields to QAMessage methods used to get and set message properties.

For example, assume you have the following QAMessage instance:

```
QAMessage msg = mgr.createTextMessage();
```

The following example gets the value corresponding to MessageProperties.MSG_TYPE using the QAMessage.getIntProperty(String) method. The MessageType enumeration maps the integer result to an appropriate message type.

```
int msg_type = t_msg.getIntProperty( MessageProperties.MSG_TYPE );
```

The following example shows the onSystemMessage(QAMessage) method, which is used to handle QAnywhere system messages.

The message type is evaluated using MessageProperties.MSG_TYPE variable and the QAMessage.getIntProperty(String) method.

A delimited list of RAS entry names is obtained using MessageProperties.RASNAMES and the QAMessage.getStringProperty(String) method.

```
private void onSystemMessage(QAMessage msg) {
  QATextMessage    t_msg;
  int              msg_type;
  String           network_adapters;
  String           network_names;
  String           network_info;

  t_msg = (QATextMessage)msg;
  if( t_msg != null ) {
      // Evaluate the message type.
      msg_type =
(MessageType)t_msg.getIntProperty( MessageProperties.MSG_TYPE );
      if( msg_type == MessageType.NETWORK_STATUS_NOTIFICATION ) {
          // Handle network status notification.
          network_info = "";
          network_adapters =
t_msg.getStringProperty( MessageProperties.ADAPTERS );
          if( network_adapters != null && network_adapters.length > 0) {
            network_info += network_adapters;
          }
          network_names =
t_msg.getStringProperty( MessageProperties.RASNAMES );

           //...
      }
  }
}
```

ADAPTER variable

For "system" queue messages, the network adapter that is being used to connect to the QAnywhere server.

Syntax

```
final String MessageProperties.ADAPTER
```

Remarks

The value of this field is "ias_Network.Adapter".

You can pass MessageProperties.ADAPTER in the QAMessage.getStringProperty(String) method to access the associated property.

This property is read-only.

See also

- "MessageProperties interface [QAnywhere Java]" on page 469
- "QAMessage.getStringProperty method [QAnywhere Java]" on page 560

ADAPTERS variable

This property name refers to a delimited list of network adapters that can be used to connect to the QAnywhere server.

Syntax

```
final String MessageProperties.ADAPTERS
```

Remarks

It is used for system queue messages.

You can pass MessageProperties.ADAPTERS in the QAMessage.getStringProperty(String) method to access the associated property. This property is read-only.

See also

- "MessageProperties interface [QAnywhere Java]" on page 469
- "QAMessage.getStringProperty method [QAnywhere Java]" on page 560

DELIVERY_COUNT variable

This property name refers to the number of attempts that have been made so far to deliver the message.

Syntax

```
final String MessageProperties.DELIVERY_COUNT
```


IP variable

For "system" queue messages, the IP address of the network adapter that is being used to connect to the QAnywhere server.

Syntax

```
final String MessageProperties.IP
```

Remarks

The value of this field is "ias_Network.IP".

You can pass MessageProperties.IP in the QAMessage.getStringProperty(String) method to access the associated property.

This property is read-only.

See also

- "MessageProperties interface [QAnywhere Java]" on page 469
- "QAMessage.getStringProperty method [QAnywhere Java]" on page 560

MAC variable

For "system" queue messages, the MAC address of the network adapter that is being used to connect to the QAnywhere server.

Syntax

```
final String MessageProperties.MAC
```

Remarks

The value of this field is "ias_Network.MAC".

You can pass MessageProperties.MAC in the QAMessage.getStringProperty(String) method to access the associated property.

This property is read-only.

See also

- "MessageProperties interface [QAnywhere Java]" on page 469
- "QAMessage.getStringProperty method [QAnywhere Java]" on page 560

MSG_TYPE variable

This property name refers to MessageType enumeration values associated with a QAnywhere message.

Syntax

```
final String MessageProperties.MSG_TYPE
```

Remarks

The value of this field is "ias_MessageType".

You can pass MessageProperties.MSG_TYPE in the QAMessage.getIntProperty(String) method to access the associated property.

This property is read-only.

See also

- "MessageProperties interface [QAnywhere Java]" on page 469
- "QAMessage.getIntProperty method [QAnywhere Java]" on page 556

NETWORK_STATUS variable

This property name refers to the state of the network connection.

Syntax

```
final String MessageProperties.NETWORK_STATUS
```

Remarks

The value is 1 if the network is accessible and 0 otherwise. The network status is used for system queue messages (for example, network status changes).

You can pass MessageProperties.NETWORK_STATUS in the QAMessage.getIntProperty(String) method to access the associated property.

This property is read-only.

See also

- "MessageProperties interface [QAnywhere Java]" on page 469
- "QAMessage.getIntProperty method [QAnywhere Java]" on page 556

ORIGINATOR variable

This property name refers to the message store ID of the originator of the message.

Syntax

```
final String MessageProperties.ORIGINATOR
```


RAS variable

For "system" queue messages, the RAS entry name that is being used to connect to the QAnywhere server.

Syntax

```
final String MessageProperties.RAS
```

Remarks

The value of this field is "ias_Network.RAS".

You can pass MessageProperties.RAS in the QAMessage.getStringProperty(String) method to access the associated property.

This property is read-only.

See also

- “MessageProperties interface [QAnywhere Java]” on page 469
- “QAMessage.getStringProperty method [QAnywhere Java]” on page 560

RASNAMES variable

For "system" queue messages, a delimited list of RAS entry names that can be used to connect to the QAnywhere server.

Syntax

```
final String MessageProperties.RASNAMES
```

Remarks

The value of this field is "ias_RASNames".

You can pass MessageProperties.RASNAMES in the QAMessage.getStringProperty(String) method to access the associated property.

This property is read-only.

See also

- “MessageProperties interface [QAnywhere Java]” on page 469
- “QAMessage.getStringProperty method [QAnywhere Java]” on page 560

STATUS variable

This property name refers to the current status of the message.

Syntax

```
final String MessageProperties.STATUS
```

See also

- "StatusCodes interface [QAnywhere Java]" on page 581

STATUS_TIME variable

This property name refers to the time at which the message assumed its current status.

Syntax

```
final String MessageProperties.STATUS_TIME
```

Remarks

If you pass MessageProperties.STATUS_TIME to the QAMessage.getProperty method, it returns a java.util.Date instance.

See also

- "QAMessage.getProperty method [QAnywhere Java]" on page 557

TRANSMISSION_STATUS variable

This property name refers to the current transmission status of the message.

Syntax

```
final String MessageProperties.TRANSMISSION_STATUS
```

See also

- "StatusCodes interface [QAnywhere Java]" on page 581

MessageStoreProperties interface

This class defines constant values for useful message store property names.

Syntax

```
public interface MessageStoreProperties
```

Members

All members of MessageStoreProperties interface, including all inherited members.

Name	Description
MAX_DELIVERY_ATTEMPTS variable	This property name refers to the maximum number of times that a message can be received without being acknowledged before its status is set to StatusCodes.UNRECEIVABLE.

Remarks

The MessageStoreProperties class provides standard message property names. You can pass MessageStoreProperties fields to QAManagerBase methods used to get and set predefined or custom message store properties.

See also

- "QAManagerBase interface [QAnywhere Java]" on page 514

MAX_DELIVERY_ATTEMPTS variable

This property name refers to the maximum number of times that a message can be received without being acknowledged before its status is set to StatusCodes.UNRECEIVABLE.

Syntax

```
final String MessageStoreProperties.MAX_DELIVERY_ATTEMPTS
```

See also

- "StatusCodes.UNRECEIVABLE variable [QAnywhere Java]" on page 585

MessageType interface

Defines constant values for the MessageProperties.MSG_TYPE message property.

Syntax

```
public interface MessageType
```

Members

All members of MessageType interface, including all inherited members.

Name	Description
NETWORK_STATUS_NOTIFICATION variable	Identifies a QAnywhere system message used to notify QAnywhere client applications of network status changes.
PUSH_NOTIFICATION variable	Identifies a QAnywhere system message used to notify QAnywhere client applications of push notifications.

Name	Description
REGULAR variable	If no message type property exists, the message type is assumed to be REGULAR.

Remarks

The following example shows the onSystemMessage(QAMessage) method, which is used to handle QAnywhere system messages. The message type is compared to MessageType.NETWORK_STATUS_NOTIFICATION.

```
private void onSystemMessage(QAMessage msg)
{
  QATextMessage    t_msg;
  int              msg_type;
  String           network_adapters;
  String           network_names;
  String           network_info;

  t_msg = (QATextMessage)msg;
  if( t_msg != null )
  {
      // Evaluate message type.
      msg_type = t_msg.getIntProperty( MessageProperties.MSG_TYPE );
      if( msg_type == MessageType.NETWORK_STATUS_NOTIFICATION )
      {
          // Handle network status notification.

      }
  }
}
```

NETWORK_STATUS_NOTIFICATION variable

Identifies a QAnywhere system message used to notify QAnywhere client applications of network status changes.

Syntax

```
final int MessageType.NETWORK_STATUS_NOTIFICATION
```

Remarks

Network status changes apply to the device receiving the system message. Use the MessageProperties.ADAPTER, MessageProperties.NETWORK, and MessageProperties.NETWORK_STATUS fields to identify new network status information.

PUSH_NOTIFICATION variable

Identifies a QAnywhere system message used to notify QAnywhere client applications of push notifications.

Syntax

```
final int MessageType.PUSH_NOTIFICATION
```

Remarks

If you use the on-demand QAnywhere Agent policy, a typical response is to call the QAManagerBase.triggerSendReceive() method to receive messages waiting with the central message server.

REGULAR variable

If no message type property exists, the message type is assumed to be REGULAR.

Syntax

```
final int MessageType.REGULAR
```

Remarks

This type of message is not treated specially by the message system.

PropertyType interface

QAMessage property type enumeration, corresponding naturally to the Java types.

Syntax

```
public interface PropertyType
```

Members

All members of PropertyType interface, including all inherited members.

Name	Description
PROPERTY_TYPE_BOOLEAN variable	Indicates a boolean property.
PROPERTY_TYPE_BYTE variable	Indicates a signed byte property.
PROPERTY_TYPE_DOUBLE variable	Indicates a double property.
PROPERTY_TYPE_FLOAT variable	Indicates a float property.
PROPERTY_TYPE_INT variable	Indicates an int property.
PROPERTY_TYPE_LONG variable	Indicates an long property.
PROPERTY_TYPE_SHORT variable	Indicates a short property.

Name	Description
PROPERTY_TYPE_STRING variable	Indicates a String property.
PROPERTY_TYPE_UNKNOWN variable	Indicates an unknown property type, usually because the property is unknown.

See also

- "QAMessage interface [QAnywhere Java]" on page 550

PROPERTY_TYPE_BOOLEAN variable

Indicates a boolean property.

Syntax

```
final short PropertyType.PROPERTY_TYPE_BOOLEAN
```

PROPERTY_TYPE_BYTE variable

Indicates a signed byte property.

Syntax

```
final short PropertyType.PROPERTY_TYPE_BYTE
```

PROPERTY_TYPE_DOUBLE variable

Indicates a double property.

Syntax

```
final short PropertyType.PROPERTY_TYPE_DOUBLE
```

PROPERTY_TYPE_FLOAT variable

Indicates a float property.

Syntax

```
final short PropertyType.PROPERTY_TYPE_FLOAT
```


PROPERTY_TYPE_INT variable

Indicates an int property.

Syntax

```
final short PropertyType.PROPERTY_TYPE_INT
```

PROPERTY_TYPE_LONG variable

Indicates an long property.

Syntax

```
final short PropertyType.PROPERTY_TYPE_LONG
```

PROPERTY_TYPE_SHORT variable

Indicates a short property.

Syntax

```
final short PropertyType.PROPERTY_TYPE_SHORT
```

PROPERTY_TYPE_STRING variable

Indicates a String property.

Syntax

```
final short PropertyType.PROPERTY_TYPE_STRING
```

PROPERTY_TYPE_UNKNOWN variable

Indicates an unknown property type, usually because the property is unknown.

Syntax

```
final short PropertyType.PROPERTY_TYPE_UNKNOWN
```

QABinaryMessage interface

A QABinaryMessage object is used to send a message containing a stream of uninterpreted bytes.

Syntax

```
public interface QABinaryMessage
```

Base classes

- "QAMessage interface [QAnywhere Java]" on page 550

Members

All members of QABinaryMessage interface, including all inherited members.

Name	Description
clearProperties method	Clear all the properties of the message.
getAddress method	Returns the destination address for the QAMessage instance.
getBodyLength method	Returns the size of the message body in bytes.
getBooleanProperty method	Gets a boolean message property.
getByteProperty method	Gets a signed byte message property.
getDoubleProperty method	Gets a double message property.
getExpiration method	Returns the message's expiration value, or null if the message does not expire or has not yet been sent.
getFloatProperty method	Gets a float message property.
getInReplyToID method	Returns the message ID of the message to which this message is a reply.
getIntProperty method	Gets an int message property.
getLongProperty method	Gets a long message property.
getMessageID method	Returns the globally unique message ID of the message.
getPriority method	Returns the priority of the message (ranging from 0 to 9).
getProperty method	Gets a message property.
getPropertyNames method	Gets an enumerator over the property names of the message.
getPropertyType method	Returns the property type of the given property.
getRedelivered method	Indicates whether the message has been previously received but not acknowledged.
getReplyToAddress method	Returns the reply-to address of this message.

Name	Description
getShortProperty method	Gets a short message property.
getStringProperty method	Gets a String message property.
getTimestamp method	Returns the message timestamp, which is the time the message was created.
propertyExists method	Indicates whether the given property has been set for this message.
readBinary method	Reads some number of bytes starting from the unread portion of a QABinaryMessage instance body and stores them into the **dest** array.
readBoolean method	Reads a boolean value starting from the unread portion of the QABinaryMessage instance's message body.
readByte method	Reads a signed byte value starting from the unread portion of a QABinaryMessage message body.
readChar method	Reads a char value starting from the unread portion of a QABinaryMessage message body.
readDouble method	Reads a double value starting from the unread portion of a QABinaryMessage message body.
readFloat method	Reads a float value starting from the unread portion of a QABinaryMessage message body.
readInt method	Reads an integer value starting from the unread portion of a QABinaryMessage message body.
readLong method	Reads a long value starting from the unread portion of a QABinaryMessage message body.
readShort method	Reads a short value starting from the unread portion of a QABinaryMessage message body.
readString method	Reads a string value starting from the unread portion of a QABinaryMessage message body.
reset method	Resets a message so that the reading of values starts from the beginning of the message body.
setBooleanProperty method	Sets a boolean property.
setByteProperty method	Sets a signed byte property.

Name	Description
setDoubleProperty method	Sets a double property.
setFloatProperty method	Sets a float property.
setInReplyToID method	Sets the in reply to ID, which identifies the message this message is a reply to.
setIntProperty method	Sets an int property.
setLongProperty method	Sets a long property.
setPriority method	Sets the priority of the message (ranging from 0 to 9).
setProperty method	Sets a property.
setReplyToAddress method	Sets the reply-to address.
setShortProperty method	Sets a short property.
setStringProperty method	Sets a string property.
writeBinary method	Appends a byte array value to the QABinaryMessage instance's message body.
writeBoolean method	Appends a boolean value to the QABinaryMessage instance's message body.
writeByte method	Appends a signed byte value to the QABinaryMessage instance's message body.
writeChar method	Appends a char value to the QABinaryMessage instance's message body.
writeDouble method	Appends a double value to the QABinaryMessage instance's message body.
writeFloat method	Appends a float value to the QABinaryMessage instance's message body.
writeInt method	Appends an integer value to the QABinaryMessage instance's message body.
writeLong method	Appends a long value to the QABinaryMessage instance's message body.
writeShort method	Appends a short value to the QABinaryMessage instance's message body.

Name	Description
writeString method	Appends a string value to the QABinaryMessage instance's message body.
DEFAULT_PRIORITY variable	The default message priority.
DEFAULT_TIME_TO_LIVE variable	The default time-to-live value.

Remarks

QABinaryMessage inherits from the QAMessage class and adds a bytes message body. QABinaryMessage provides a variety of functions to read from and write to the bytes message body.

When the message is first created, the body of the message is in write-only mode. After a message has been sent, the client that sent it can retain and modify it without affecting the message that has been sent. The same message object can be sent multiple times.

When a message is received, the provider has called QABinaryMessage.reset() so that the message body is in read-only mode and reading of values starts from the beginning of the message body.

The following example uses the QABinaryMessage.writeString(String) to write the string "Q" followed by the string "Anywhere" to a QABinaryMessage instance's message body.

```
// Create a binary message instance.
QABinaryMessage binary_message;
binary_message = qa_manager.createBinaryMessage();

// Set optional message properties.
binary_message.setReplyToAddress("my-queue-name");

// Write to the message body.
binary_message.writeString("Q");
binary_message.writeString("Anywhere");

// Put the message in the local database, ready for sending.
try {
    qa_manager.putMessage( "store-id\\queue-name", binary_message );
}
catch ( QAException e ) {
    handleError();
}
```

Note
On the receiving end, the first QABinaryMessage.readString() invocation returns "Q" and the next QABinaryMessage.readString() invocation returns "Anywhere".

The message is sent by the QAnywhere Agent.

See also

- "QAMessage interface [QAnywhere Java]" on page 550
- "QABinaryMessage.readString method [QAnywhere Java]" on page 491

getBodyLength method

Returns the size of the message body in bytes.

Syntax

```
long QABinaryMessage.getBodyLength() throws QAException
```

Returns

The size of the message body in bytes.

Exceptions

- **QAException class** Thrown if there is a problem retrieving the size of the message body.

readBinary method

Reads some number of bytes starting from the unread portion of a QABinaryMessage instance body and stores them into the dest array.

Overload list

Name	Description
readBinary(byte[]) method	Reads some number of bytes starting from the unread portion of a QABinaryMessage instance body and stores them into the **dest** array.
readBinary(byte[], int) method	Reads up to length number of bytes starting from the unread portion of a QABinaryMessage instance body and stores them into the **dest** array.
readBinary(byte[], int, int) method	Reads up to length number of bytes starting from the unread portion of a QABinaryMessage instance body and stores them into the **dest** array starting at dest[offset].

readBinary(byte[]) method

Reads some number of bytes starting from the unread portion of a QABinaryMessage instance body and stores them into the **dest** array.

Syntax

```
int QABinaryMessage.readBinary(byte[] dest) throws QAException
```

Parameters

- **dest** The byte array to hold the read bytes.

Returns

The number of bytes read from the message body, or -1 if there are no more bytes available.

Exceptions

- **QAException class** Thrown if there was an error reading bytes from the message.

Remarks

The readBinary(dest) method has the same effect as: readBinary(dest,0,dest.length)

See also

- "QABinaryMessage.writeBinary method [QAnywhere Java]" on page 492

readBinary(byte[], int) method

Reads up to length number of bytes starting from the unread portion of a QABinaryMessage instance body and stores them into the **dest** array.

Syntax

```
int QABinaryMessage.readBinary(
     byte[] dest,
    int length
) throws QAException
```

Parameters

- **dest** The byte array to hold the read bytes.
- **length** The maximum number of bytes to read.

Returns

The number of bytes read from the message body, or -1 if there are no more bytes available.

Exceptions

- **QAException class** Thrown if there was an error reading bytes from the message.

Remarks

The readBinary(dest, len) method has the same effect as: readBinary(dest,0,len)

See also

- "QABinaryMessage.writeBinary method [QAnywhere Java]" on page 492

readBinary(byte[], int, int) method

Reads up to length number of bytes starting from the unread portion of a QABinaryMessage instance body and stores them into the **dest** array starting at dest[offset].

Syntax

```
int QABinaryMessage.readBinary(
     byte[] dest,
    int offset,
```

```
    int length
) throws QAException
```

Parameters

- **dest** The byte array to hold the read bytes.
- **offset** The start offset of the destination array.
- **length** The maximum number of bytes to read.

Returns

The number of bytes read from the message body, or -1 if there are no more bytes available.

Exceptions

- **QAException class** Thrown if there was an error reading bytes from the message.

Remarks

If dest is null, a NullPointerException is thrown.

If offset is negative, or length is negative, or offset+length is greater than the length of dest, then an IndexOutOfBoundsException is thrown.

See also

- "QABinaryMessage.writeBinary method [QAnywhere Java]" on page 492

readBoolean method

Reads a boolean value starting from the unread portion of the QABinaryMessage instance's message body.

Syntax

```
boolean QABinaryMessage.readBoolean() throws QAException
```

Returns

The boolean value read from the message body.

Exceptions

- **QAException class** Thrown if there was a conversion error reading the value or if there is no more input.

See also

- "QABinaryMessage.writeBoolean method [QAnywhere Java]" on page 494

readByte method

Reads a signed byte value starting from the unread portion of a QABinaryMessage message body.

Syntax

```
byte QABinaryMessage.readByte() throws QAException
```

Returns

The signed byte value read from the message body.

Exceptions

- **QAException class** Thrown if there was a conversion error reading the value or if there is no more input.

See also

- "QABinaryMessage.writeByte method [QAnywhere Java]" on page 494

readChar method

Reads a char value starting from the unread portion of a QABinaryMessage message body.

Syntax

```
char QABinaryMessage.readChar() throws QAException
```

Returns

The character value read from the message body.

Exceptions

- **QAException class** Thrown if there was a conversion error reading the value or if there is no more input.

See also

- "QABinaryMessage.writeChar method [QAnywhere Java]" on page 495

readDouble method

Reads a double value starting from the unread portion of a QABinaryMessage message body.

Syntax

```
double QABinaryMessage.readDouble() throws QAException
```

Returns

The double value read from the message body.

Exceptions

- **QAException class** Thrown if there was a conversion error reading the value or if there is no more input.

See also

- "QABinaryMessage.writeDouble method [QAnywhere Java]" on page 495

readFloat method

Reads a float value starting from the unread portion of a QABinaryMessage message body.

Syntax

```
float QABinaryMessage.readFloat() throws QAException
```

Returns

The float value read from the message body.

Exceptions

- **QAException class** Thrown if there was a conversion error reading the value or if there is no more input.

See also

- "QABinaryMessage.writeFloat method [QAnywhere Java]" on page 496

readInt method

Reads an integer value starting from the unread portion of a QABinaryMessage message body.

Syntax

```
int QABinaryMessage.readInt() throws QAException
```

Returns

The int value read from the message body.

Exceptions

- **QAException class** Thrown if there was a conversion error reading the value or if there is no more input.

See also

- "QABinaryMessage.writeInt method [QAnywhere Java]" on page 496

readLong method

Reads a long value starting from the unread portion of a QABinaryMessage message body.

Syntax

```
long QABinaryMessage.readLong() throws QAException
```

Returns

The long value read from the message body.

Exceptions

- **QAException class** Thrown if there was a conversion error reading the value or if there is no more input.

See also

- "QABinaryMessage.writeLong method [QAnywhere Java]" on page 497

readShort method

Reads a short value starting from the unread portion of a QABinaryMessage message body.

Syntax

```
short QABinaryMessage.readShort() throws QAException
```

Returns

The short value read from the message body.

Exceptions

- **QAException class** Thrown if there was a conversion error reading the value or if there is no more input.

See also

- "QABinaryMessage.writeShort method [QAnywhere Java]" on page 497

readString method

Reads a string value starting from the unread portion of a QABinaryMessage message body.

Syntax

```
String QABinaryMessage.readString() throws QAException
```

Returns

The string value read from the message body.

Exceptions

- **QAException class** Thrown if there was a conversion error reading the value or if there is no more input.

See also

- "QABinaryMessage.writeString method [QAnywhere Java]" on page 498

reset method

Resets a message so that the reading of values starts from the beginning of the message body.

Syntax

```
void QABinaryMessage.reset() throws QAException
```

Exceptions

- **QAException class** Thrown if there is a problem resetting the message.

Remarks

The reset method also puts the QABinaryMessage message body in read-only mode.

writeBinary method

Appends a byte array value to the QABinaryMessage instance's message body.

Overload list

Name	Description
writeBinary(byte[]) method	Appends a byte array value to the QABinaryMessage instance's message body.
writeBinary(byte[], int) method	Appends length bytes from a byte array to the QABinaryMessage instance's message body.
writeBinary(byte[], int, int) method	Appends length bytes from a byte array starting at the given offset to the QABinaryMessage instance's message body.

writeBinary(byte[]) method

Appends a byte array value to the QABinaryMessage instance's message body.

Syntax

```
void QABinaryMessage.writeBinary(byte[] val) throws QAException
```


Parameters

- **val** The byte array value to write to the message body.

Exceptions

- **QAException class** Thrown if there is a problem appending the byte array to the message body.

Remarks

The writeBinary(val) method has the same effect as: writeBinary(val,0,val.length)

See also

- "QABinaryMessage.readBinary method [QAnywhere Java]" on page 486

writeBinary(byte[], int) method

Appends length bytes from a byte array to the QABinaryMessage instance's message body.

Syntax

```
void QABinaryMessage.writeBinary(
     byte[] val,
    int length
) throws QAException
```

Parameters

- **val** The byte array value to write to the message body.
- **length** The number of bytes to write.

Exceptions

- **QAException class** Thrown if there is a problem appending the byte array to the message body.

Remarks

The writeBinary(val,len) method has the same effect as: writeBinary(val,0,len)

See also

- "QABinaryMessage.readBinary method [QAnywhere Java]" on page 486

writeBinary(byte[], int, int) method

Appends length bytes from a byte array starting at the given offset to the QABinaryMessage instance's message body.

Syntax

```
void QABinaryMessage.writeBinary(
     byte[] val,
    int offset,
```

```
    int length
) throws QAException
```

Parameters

- **val** The byte array value to write to the message body.
- **offset** The offset within the byte array to begin writing.
- **length** The number of bytes to write.

Exceptions

- **QAException class** Thrown if there is a problem appending the byte array to the message body.

Remarks

If val is null, a NullPointerException is thrown. If offset is negative, or length is negative, or length+offset is greater than the length of val then an IndexOutOfBoundsException is thrown.

See also

- "QABinaryMessage.readBinary method [QAnywhere Java]" on page 486

writeBoolean method

Appends a boolean value to the QABinaryMessage instance's message body.

Syntax

```
void QABinaryMessage.writeBoolean(boolean val) throws QAException
```

Parameters

- **val** The boolean value to write to the message body.

Exceptions

- **QAException class** Thrown if there is a problem appending the boolean value to the message body.

Remarks

The boolean is represented as a one byte value. True is represented as 1; false is represented as 0.

See also

- "QABinaryMessage.readBoolean method [QAnywhere Java]" on page 488

writeByte method

Appends a signed byte value to the QABinaryMessage instance's message body.

Syntax

```
void QABinaryMessage.writeByte(byte val) throws QAException
```

Parameters

- **val** The signed byte value to write to the message body.

Exceptions

- **QAException class** Thrown if there is a problem appending the signed byte value to the message body.

Remarks

The signed byte is represented as a one byte value.

See also

- "QABinaryMessage.readByte method [QAnywhere Java]" on page 489

writeChar method

Appends a char value to the QABinaryMessage instance's message body.

Syntax

```
void QABinaryMessage.writeChar(char val) throws QAException
```

Parameters

- **val** The char value to write to the message body.

Exceptions

- **QAException class** Thrown if there is a problem appending the char value to the message body.

Remarks

The char is represented as a two byte value and the high order byte is appended first.

See also

- "QABinaryMessage.readChar method [QAnywhere Java]" on page 489

writeDouble method

Appends a double value to the QABinaryMessage instance's message body.

Syntax

```
void QABinaryMessage.writeDouble(double val) throws QAException
```

Parameters

- **val** the double value to write to the message body.

Exceptions

- **QAException class** Thrown if there is a problem appending the double value to the message body.

Remarks

The double is converted to a representative 8-byte long and higher order bytes are appended first.

See also

- "QABinaryMessage.readDouble method [QAnywhere Java]" on page 489

writeFloat method

Appends a float value to the QABinaryMessage instance's message body.

Syntax

```
void QABinaryMessage.writeFloat(float val) throws QAException
```

Parameters

- **val** The float value to write to the message body.

Exceptions

- **QAException class** Thrown if there is a problem appending the float value to the message body.

Remarks

The float is converted to a representative 4-byte integer and the higher order bytes are appended first.

See also

- "QABinaryMessage.readFloat method [QAnywhere Java]" on page 490

writeInt method

Appends an integer value to the QABinaryMessage instance's message body.

Syntax

```
void QABinaryMessage.writeInt(int val) throws QAException
```

Parameters

- **val** The int value to write to the message body.

Exceptions

- **QAException class** Thrown if there is a problem appending the integer value to the message body.

Remarks

The integer parameter is represented as a 4 byte value and higher order bytes are appended first.

See also

- "QABinaryMessage.readInt method [QAnywhere Java]" on page 490

writeLong method

Appends a long value to the QABinaryMessage instance's message body.

Syntax

```
void QABinaryMessage.writeLong(long val) throws QAException
```

Parameters

- **val** The long value to write to the message body.

Exceptions

- **QAException class** Thrown if there is a problem appending the long value to the message body.

Remarks

The long parameter is represented using 8-bytes value and higher order bytes are appended first.

See also

- "QABinaryMessage.readLong method [QAnywhere Java]" on page 491

writeShort method

Appends a short value to the QABinaryMessage instance's message body.

Syntax

```
void QABinaryMessage.writeShort(short val) throws QAException
```

Parameters

- **val** The short value to write to the message body.

Exceptions

- **QAException class** Thrown if there is a problem appending the short value to the message body.

Remarks

The short parameter is represented as a two byte value and the higher order byte is appended first.

See also

- "QABinaryMessage.readShort method [QAnywhere Java]" on page 491

writeString method

Appends a string value to the QABinaryMessage instance's message body.

Syntax

```
void QABinaryMessage.writeString(String val) throws QAException
```

Parameters

- **val** The string value to write to the message body.

Exceptions

- **QAException class** Thrown if there is a problem appending the string value to the message body.

Remarks

Note
The receiving application needs to invoke QABinaryMessage.readString for each writeString invocation.

Note
The UTF-8 representation of the string to be written can be at most 32767 bytes.

See also

- "QABinaryMessage.readString method [QAnywhere Java]" on page 491

QAException class

Encapsulates QAnywhere client application exceptions.

Syntax

```
public class QAException
```

Members

All members of QAException class, including all inherited members.

Name	Description
getDetailedMessage method	Returns the detailed text description of the exception.
getErrorCode method	Returns the error code of the last exception.
getNativeErrorCode method	Returns the native error code of the last exception.
COMMON_ALREADY_OPEN_ERROR variable	The QAManager is already open.
COMMON_GET_INIT_FILE_ERROR variable	Unable to access the client properties file.
COMMON_GET_PROPERTY_ERROR variable	Error retrieving property from message store.
COMMON_GETQUEUEDEPTH_ERROR variable	Error getting the queue depth.
COMMON_GETQUEUEDEPTH_ERROR_INVALID_ARG variable	Cannot use QAManagerBase.getQueueDepth on a given destination when filter is ALL.
COMMON_GETQUEUEDEPTH_ERROR_NO_STORE_ID variable	Cannot use QAManagerBase.getQueueDepth when the message store ID has not been set.
COMMON_INIT_ERROR variable	Initialization error.
COMMON_INIT_THREAD_ERROR variable	Error initializing the background thread.
COMMON_INVALID_PROPERTY variable	There is an invalid property in the client properties file.
COMMON_MSG_ACKNOWLEDGE_ERROR variable	Error acknowledging the message.
COMMON_MSG_CANCEL_ERROR variable	Error canceling message.
COMMON_MSG_CANCEL_ERROR_SENT variable	Error canceling message.
COMMON_MSG_NOT_WRITEABLE_ERROR variable	You cannot write to a message that is in read-only mode.
COMMON_MSG_RETRIEVE_ERROR variable	Error retrieving a message from the client message store.
COMMON_MSG_STORE_ERROR variable	Error storing a message in the client message store.
COMMON_MSG_STORE_NOT_INITIALIZED variable	The message store has not been initialized for messaging.

Name	Description
COMMON_MSG_STORE_TOO_LARGE variable	The message store is too large relative to the free disk space on the device.
COMMON_NO_DEST_ERROR variable	No destination.
COMMON_NO_IMPLEMENTATION variable	The method is not implemented.
COMMON_NOT_OPEN_ERROR variable	The QAManager is not open.
COMMON_OPEN_ERROR variable	Error opening a connection to the message store.
COMMON_OPEN_LOG_FILE_ERROR variable	Error opening the log file.
COMMON_OPEN_MAXTHREADS_ERROR variable	Cannot open the QAManager because the maximum number of concurrent server requests is not high enough (see database server -gn option).
COMMON_REOPEN_ERROR variable	Error re-opening connection to message store.
COMMON_SELECTOR_SYNTAX_ERROR variable	The given selector has a syntax error.
COMMON_SET_PROPERTY_ERROR variable	Error storing property to message store.
COMMON_TERMINATE_ERROR variable	Termination error.
COMMON_UNEXPECTED_EOM_ERROR variable	Unexpected end of message reached.
COMMON_UNREPRESENTABLE_TIMESTAMP variable	The timestamp is outside of the acceptable range.
QA_NO_ERROR variable	No error.

Remarks

You can use the QAException class to catch QAnywhere exceptions.

The following example catches a QAnywhere exception:

```
try {
    _qaManager = QAManagerFactory.getInstance().CreateQAManager();
    _qaManager.open( AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT );
    _qaManager.start();
}
catch( QAException e ) {
    // Handle exception.
    System.err.println("Error code: " + e.getErrorCode() );
```



```
        System.err.println("Error message: " + e.getMessage() );
    }
```

getDetailedMessage method

Returns the detailed text description of the exception.

Syntax

```
abstract String QAException.getDetailedMessage()
```

Returns

The detailed text description of the exception.

getErrorCode method

Returns the error code of the last exception.

Syntax

```
abstract int QAException.getErrorCode()
```

Returns

The error code of the last exception.

getNativeErrorCode method

Returns the native error code of the last exception.

Syntax

```
abstract int QAException.getNativeErrorCode()
```

Returns

The native error code of the last exception.

COMMON_ALREADY_OPEN_ERROR variable

The QAManager is already open.

Syntax

```
final int QAException.COMMON_ALREADY_OPEN_ERROR
```


See also

- "QAManager interface [QAnywhere Java]" on page 507

COMMON_GET_INIT_FILE_ERROR variable

Unable to access the client properties file.

Syntax

```
final int QAException.COMMON_GET_INIT_FILE_ERROR
```

COMMON_GET_PROPERTY_ERROR variable

Error retrieving property from message store.

Syntax

```
final int QAException.COMMON_GET_PROPERTY_ERROR
```

COMMON_GETQUEUEDEPTH_ERROR variable

Error getting the queue depth.

Syntax

```
final int QAException.COMMON_GETQUEUEDEPTH_ERROR
```

See also

- "QAManagerBase.getQueueDepth method [QAnywhere Java]" on page 530

COMMON_GETQUEUEDEPTH_ERROR_INVALID_ARG variable

Cannot use QAManagerBase.getQueueDepth on a given destination when filter is ALL.

Syntax

```
final int QAException.COMMON_GETQUEUEDEPTH_ERROR_INVALID_ARG
```

See also

- "QAManagerBase.getQueueDepth method [QAnywhere Java]" on page 530

COMMON_GETQUEUEDEPTH_ERROR_NO_STORE_ID variable

Cannot use QAManagerBase.getQueueDepth when the message store ID has not been set.

Syntax

```
final int QAException.COMMON_GETQUEUEDEPTH_ERROR_NO_STORE_ID
```

See also

- "QAManagerBase.getQueueDepth method [QAnywhere Java]" on page 530

COMMON_INIT_ERROR variable

Initialization error.

Syntax

```
final int QAException.COMMON_INIT_ERROR
```

COMMON_INIT_THREAD_ERROR variable

Error initializing the background thread.

Syntax

```
final int QAException.COMMON_INIT_THREAD_ERROR
```

COMMON_INVALID_PROPERTY variable

There is an invalid property in the client properties file.

Syntax

```
final int QAException.COMMON_INVALID_PROPERTY
```

COMMON_MSG_ACKNOWLEDGE_ERROR variable

Error acknowledging the message.

Syntax

```
final int QAException.COMMON_MSG_ACKNOWLEDGE_ERROR
```

COMMON_MSG_CANCEL_ERROR variable

Error canceling message.

Syntax

```
final int QAException.COMMON_MSG_CANCEL_ERROR
```

COMMON_MSG_CANCEL_ERROR_SENT variable

Error canceling message.

Syntax

```
final int QAException.COMMON_MSG_CANCEL_ERROR_SENT
```

Remarks

You cannot cancel a message that has already been sent.

COMMON_MSG_NOT_WRITEABLE_ERROR variable

You cannot write to a message that is in read-only mode.

Syntax

```
final int QAException.COMMON_MSG_NOT_WRITEABLE_ERROR
```

COMMON_MSG_RETRIEVE_ERROR variable

Error retrieving a message from the client message store.

Syntax

```
final int QAException.COMMON_MSG_RETRIEVE_ERROR
```

COMMON_MSG_STORE_ERROR variable

Error storing a message in the client message store.

Syntax

```
final int QAException.COMMON_MSG_STORE_ERROR
```

COMMON_MSG_STORE_NOT_INITIALIZED variable

The message store has not been initialized for messaging.

Syntax

```
final int QAException.COMMON_MSG_STORE_NOT_INITIALIZED
```


COMMON_MSG_STORE_TOO_LARGE variable

The message store is too large relative to the free disk space on the device.

Syntax

```
final int QAException.COMMON_MSG_STORE_TOO_LARGE
```

COMMON_NO_DEST_ERROR variable

No destination.

Syntax

```
final int QAException.COMMON_NO_DEST_ERROR
```

COMMON_NO_IMPLEMENTATION variable

The method is not implemented.

Syntax

```
final int QAException.COMMON_NO_IMPLEMENTATION
```

COMMON_NOT_OPEN_ERROR variable

The QAManager is not open.

Syntax

```
final int QAException.COMMON_NOT_OPEN_ERROR
```

See also

- "QAManager interface [QAnywhere Java]" on page 507

COMMON_OPEN_ERROR variable

Error opening a connection to the message store.

Syntax

```
final int QAException.COMMON_OPEN_ERROR
```

COMMON_OPEN_LOG_FILE_ERROR variable

Error opening the log file.

Syntax

```
final int QAException.COMMON_OPEN_LOG_FILE_ERROR
```

COMMON_OPEN_MAXTHREADS_ERROR variable

Cannot open the QAManager because the maximum number of concurrent server requests is not high enough (see database server -gn option).

Syntax

```
final int QAException.COMMON_OPEN_MAXTHREADS_ERROR
```

COMMON_REOPEN_ERROR variable

Error re-opening connection to message store.

Syntax

```
final int QAException.COMMON_REOPEN_ERROR
```

COMMON_SELECTOR_SYNTAX_ERROR variable

The given selector has a syntax error.

Syntax

```
final int QAException.COMMON_SELECTOR_SYNTAX_ERROR
```

COMMON_SET_PROPERTY_ERROR variable

Error storing property to message store.

Syntax

```
final int QAException.COMMON_SET_PROPERTY_ERROR
```

COMMON_TERMINATE_ERROR variable

Termination error.

Syntax

```
final int QAException.COMMON_TERMINATE_ERROR
```

COMMON_UNEXPECTED_EOM_ERROR variable

Unexpected end of message reached.

Syntax

```
final int QAException.COMMON_UNEXPECTED_EOM_ERROR
```

COMMON_UNREPRESENTABLE_TIMESTAMP variable

The timestamp is outside of the acceptable range.

Syntax

```
final int QAException.COMMON_UNREPRESENTABLE_TIMESTAMP
```

QA_NO_ERROR variable

No error.

Syntax

```
final int QAException.QA_NO_ERROR
```

QAManager interface

QAManager manages non-transactional QAnywhere messaging operations.

Syntax

```
public interface QAManager
```

Base classes

- "QAManagerBase interface [QAnywhere Java]" on page 514

Members

All members of QAManager interface, including all inherited members.

Name	Description
acknowledge method	Acknowledges that the client application successfully received a QAnywhere message.
acknowledgeAll method	Acknowledges that the client application successfully received QAnywhere messages.
acknowledgeUntil method	Acknowledges the given QAMessage instance and all unacknowledged messages received before the given message.
browseMessages method	Browses all available messages in the message store.
browseMessagesByID method	Browse the message with the given message ID.
browseMessagesByQueue method	Browses the available messages waiting that have been sent to the given address.
browseMessagesBySelector method	Browse messages queued in the message store that satisfy the given selector.
cancelMessage method	Cancels the message with the given message ID.
close method	Closes the connection to the QAnywhere message system and releases any resources used by the QAManagerBase.
createBinaryMessage method	Creates a QABinaryMessage object.
createTextMessage method	Creates a QATextMessage object.
getBooleanStoreProperty method	Gets a boolean value for a predefined or custom message store property.
getByteStoreProperty method	Gets a signed byte value for a predefined or custom message store property.
getDoubleStoreProperty method	Gets a double value for a predefined or custom message store property.
getFloatStoreProperty method	Gets a float value for a predefined or custom message store property.
getIntStoreProperty method	Gets an int value for a predefined or custom message store property.
getLongStoreProperty method	Gets a long value for a predefined or custom message store property.

Name	Description
getMessage method	Returns the next available QAMessage sent to the specified address.
getMessageBySelector method	Returns the next available QAMessage sent to the specified address that satisfies the given selector.
getMessageBySelectorNoWait method	Returns the next available QAMessage sent to the given address that satisfies the given selector.
getMessageBySelectorTimeout method	Returns the next available QAMessage sent to the given address that satisfies the given selector.
getMessageNoWait method	Returns the next available QAMessage sent to the given address.
getMessageTimeout method	Returns the next available QAMessage sent to the given address.
getMode method	Returns the QAManager acknowledgement mode for received messages.
getQueueDepth method	Returns the total depth of all queues, based on a given filter.
getShortStoreProperty method	Gets a short value for a predefined or custom message store property.
getStoreProperty method	Gets an Object representing a message store property.
getStorePropertyNames method	Gets an enumerator over the message store property names.
getStringStoreProperty method	Gets a string value for a predefined or custom message store property.
open method	Opens the QAManager with the given AcknowledgementMode value.
propertyExists method	Tests if there currently exists a value for the given the property.
putMessage method	Prepares a message to send to another QAnywhere client.
putMessageTimeToLive method	Prepares a message to send to another QAnywhere client.
recover method	Forces all unacknowledged messages into a status of StatusCodes.PENDING.
reOpen method	Reopens the QAManagerBase.
setBooleanStoreProperty method	Sets a predefined or custom message store property to a boolean value.

Name	Description
setByteStoreProperty method	Sets a predefined or custom message store property to a sbyte value.
setDoubleStoreProperty method	Sets a predefined or custom message store property to a double value.
setFloatStoreProperty method	Sets a predefined or custom message store property to a float value.
setIntStoreProperty method	Sets a predefined or custom message store property to an int value.
setLongStoreProperty method	Sets a predefined or custom message store property to a long value.
setMessageListener method	Registers a QAMessageListener object to receive QAnywhere messages asynchronously.
setMessageListener2 method	Registers a QAMessageListener2 object to receive QAnywhere messages asynchronously.
setMessageListenerBySelector method	Registers a QAMessageListener object to receive QAnywhere messages asynchronously, with a message selector.
setMessageListenerBySelector2 method	Registers a QAMessageListener2 object to receive QAnywhere messages asynchronously, with a message selector.
setProperty method	Allows you to set QAnywhere Manager configuration properties programmatically.
setShortStoreProperty method	Sets a predefined or custom message store property to a short value.
setStoreProperty method	Sets a predefined or custom message store property to a System.Object value.
setStringStoreProperty method	Sets a predefined or custom message store property to a String value.
start method	Starts the QAManagerBase for receiving incoming messages.
stop method	Halts the QAManagerBase's reception of incoming messages.
triggerSendReceive method	Causes a synchronization with the QAnywhere message server, uploading any messages addressed to other clients, and downloading any messages addressed to the local client.

Remarks

It derives from QAManagerBase.

For a detailed description of derived behavior, see QAManagerBase.

The QAManager instance can be configured for implicit or explicit acknowledgement, as defined in the AcknowledgementMode class. To acknowledge messages as part of a transaction, use QATransactionalManager.

Use the QAManagerFactory class to create QAManager and QATransactionalManager objects.

See also

- "AcknowledgementMode interface [QAnywhere Java]" on page 468
- "QAManagerFactory class [QAnywhere Java]" on page 545
- "QATransactionalManager interface [QAnywhere Java]" on page 575

acknowledge method

Acknowledges that the client application successfully received a QAnywhere message.

Syntax

```
void QAManager.acknowledge(QAMessage msg) throws QAException
```

Parameters

- **msg** The message to acknowledge.

Exceptions

- **QAException class** Thrown if there is a problem acknowledging the message.

Remarks

> **Note**
> When a QAMessage is acknowledged, its status property changes to StatusCodes.RECEIVED. It can then be deleted using the default delete rule.

See also

- "StatusCodes.RECEIVED variable [QAnywhere Java]" on page 584
- "QAManager.acknowledgeUntil method [QAnywhere Java]" on page 512
- "QAManager.acknowledgeAll method [QAnywhere Java]" on page 511

acknowledgeAll method

Acknowledges that the client application successfully received QAnywhere messages.

Syntax

```
void QAManager.acknowledgeAll() throws QAException
```

Exceptions

- **QAException class** Thrown if there is a problem acknowledging the messages.

Remarks

All unacknowledged messages are acknowledged.

> **Note**
> When a QAMessage is acknowledged, its status property changes to StatusCodes.RECEIVED. It can then be deleted using the default delete rule.

See also

- "StatusCodes.RECEIVED variable [QAnywhere Java]" on page 584
- "QAManager.acknowledge method [QAnywhere Java]" on page 511
- "QAManager.acknowledgeUntil method [QAnywhere Java]" on page 512

acknowledgeUntil method

Acknowledges the given QAMessage instance and all unacknowledged messages received before the given message.

Syntax

```
void QAManager.acknowledgeUntil(QAMessage msg) throws QAException
```

Parameters

- **msg** The last message to acknowledge. All earlier unacknowledged messages are also acknowledged.

Exceptions

- **QAException class** Thrown if there is a problem acknowledging the messages.

Remarks

> **Note**
> When a QAMessage is acknowledged, its status property changes to StatusCodes.RECEIVED. It can then be deleted using the default delete rule.

See also

- "QAMessage interface [QAnywhere Java]" on page 550
- "StatusCodes.RECEIVED variable [QAnywhere Java]" on page 584
- "QAManager.acknowledge method [QAnywhere Java]" on page 511
- "QAManager.acknowledgeAll method [QAnywhere Java]" on page 511

open method

Opens the QAManager with the given AcknowledgementMode value.

Syntax

```
void QAManager.open(short mode) throws QAException
```

Parameters

- **mode** The acknowledgement mode, one of AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT or AcknowledgementMode.IMPLICIT_ACKNOWLEDGEMENT.

Exceptions

- **QAException class** Thrown if there is a problem opening the QAManager instance.

Remarks

The open(short) method must be the first method called after creating a QAManager.

If a database connection error is detected, you can re-open a QAManager by calling the close method followed by the open method. When re-opening a QAManager, you do not need to re-create it, reset the properties, or reset the message listeners. The properties of the QAManager cannot be changed after the first open, and subsequent open calls must supply the same acknowledgement mode.

See also

- "AcknowledgementMode interface [QAnywhere Java]" on page 468
- "AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT variable [QAnywhere Java]" on page 469
- "AcknowledgementMode.IMPLICIT_ACKNOWLEDGEMENT variable [QAnywhere Java]" on page 469

recover method

Forces all unacknowledged messages into a status of StatusCodes.PENDING.

Syntax

```
void QAManager.recover() throws QAException
```

Exceptions

- **QAException class** Thrown if there is a problem recovering.

Remarks

These messages must be received again using QAManagerBase.getMessage(String).

See also

- "QAManagerBase.getMessage method [QAnywhere Java]" on page 525

QAManagerBase interface

This class acts as a base class for QATransactionalManager and QAManager, which manage transactional and non-transactional messaging, respectively.

Syntax

```
public interface QAManagerBase
```

Derived classes

- "QAManager interface [QAnywhere Java]" on page 507
- "QATransactionalManager interface [QAnywhere Java]" on page 575

Members

All members of QAManagerBase interface, including all inherited members.

Name	Description
browseMessages method	Browses all available messages in the message store.
browseMessagesByID method	Browse the message with the given message ID.
browseMessagesByQueue method	Browses the available messages waiting that have been sent to the given address.
browseMessagesBySelector method	Browse messages queued in the message store that satisfy the given selector.
cancelMessage method	Cancels the message with the given message ID.
close method	Closes the connection to the QAnywhere message system and releases any resources used by the QAManagerBase.
createBinaryMessage method	Creates a QABinaryMessage object.
createTextMessage method	Creates a QATextMessage object.
getBooleanStoreProperty method	Gets a boolean value for a predefined or custom message store property.
getByteStoreProperty method	Gets a signed byte value for a predefined or custom message store property.

Name	Description
getDoubleStoreProperty method	Gets a double value for a predefined or custom message store property.
getFloatStoreProperty method	Gets a float value for a predefined or custom message store property.
getIntStoreProperty method	Gets an int value for a predefined or custom message store property.
getLongStoreProperty method	Gets a long value for a predefined or custom message store property.
getMessage method	Returns the next available QAMessage sent to the specified address.
getMessageBySelector method	Returns the next available QAMessage sent to the specified address that satisfies the given selector.
getMessageBySelectorNoWait method	Returns the next available QAMessage sent to the given address that satisfies the given selector.
getMessageBySelectorTimeout method	Returns the next available QAMessage sent to the given address that satisfies the given selector.
getMessageNoWait method	Returns the next available QAMessage sent to the given address.
getMessageTimeout method	Returns the next available QAMessage sent to the given address.
getMode method	Returns the QAManager acknowledgement mode for received messages.
getQueueDepth method	Returns the total depth of all queues, based on a given filter.
getShortStoreProperty method	Gets a short value for a predefined or custom message store property.
getStoreProperty method	Gets an Object representing a message store property.
getStorePropertyNames method	Gets an enumerator over the message store property names.
getStringStoreProperty method	Gets a string value for a predefined or custom message store property.
propertyExists method	Tests if there currently exists a value for the given the property.
putMessage method	Prepares a message to send to another QAnywhere client.

Name	Description
putMessageTimeToLive method	Prepares a message to send to another QAnywhere client.
reOpen method	Reopens the QAManagerBase.
setBooleanStoreProperty method	Sets a predefined or custom message store property to a boolean value.
setByteStoreProperty method	Sets a predefined or custom message store property to a sbyte value.
setDoubleStoreProperty method	Sets a predefined or custom message store property to a double value.
setFloatStoreProperty method	Sets a predefined or custom message store property to a float value.
setIntStoreProperty method	Sets a predefined or custom message store property to an int value.
setLongStoreProperty method	Sets a predefined or custom message store property to a long value.
setMessageListener method	Registers a QAMessageListener object to receive QAnywhere messages asynchronously.
setMessageListener2 method	Registers a QAMessageListener2 object to receive QAnywhere messages asynchronously.
setMessageListenerBySelector method	Registers a QAMessageListener object to receive QAnywhere messages asynchronously, with a message selector.
setMessageListenerBySelector2 method	Registers a QAMessageListener2 object to receive QAnywhere messages asynchronously, with a message selector.
setProperty method	Allows you to set QAnywhere Manager configuration properties programmatically.
setShortStoreProperty method	Sets a predefined or custom message store property to a short value.
setStoreProperty method	Sets a predefined or custom message store property to a System.Object value.
setStringStoreProperty method	Sets a predefined or custom message store property to a String value.
start method	Starts the QAManagerBase for receiving incoming messages.

Name	Description
stop method	Halts the QAManagerBase's reception of incoming messages.
triggerSendReceive method	Causes a synchronization with the QAnywhere message server, uploading any messages addressed to other clients, and downloading any messages addressed to the local client.

Remarks

Use the QAManagerBase.start() method to allow a QAManagerBase instance to listen for messages. An instance of QAManagerBase must be used only on the thread that created it.

You can use instances of this class to create and manage QAnywhere messages. Use the QAManagerBase.createBinaryMessage() and QAManagerBase.createTextMessage() methods to create appropriate QAMessage instances. QAMessage instances provide a variety of methods to set message content and properties. To send QAnywhere messages, use the QAManagerBase.putMessage(String, QAMessage) method to place the addressed message in the local message store queue. The message is transmitted by the QAnywhere Agent based on its transmission policies or when you call QAManagerBase.triggerSendReceive().

QAManagerBase also provides methods to set and get message store properties.

See also

- "QATransactionalManager interface [QAnywhere Java]" on page 575
- "QAManager interface [QAnywhere Java]" on page 507

browseMessages method

Browses all available messages in the message store.

Syntax

```
java.util.Enumeration QAManagerBase.browseMessages() throws QAException
```

Returns

An enumerator over the available messages.

Exceptions

- **QAException class** Thrown if there is a problem browsing the messages.

Remarks

The messages are just being browsed, so they cannot be acknowledged.

Use the QAManagerBase.getMessage(String) method to receive messages so that they can be acknowledged.

See also

- "QAManagerBase.browseMessagesByQueue method [QAnywhere Java]" on page 518
- "QAManagerBase.browseMessagesByID method [QAnywhere Java]" on page 518
- "QAManagerBase.getMessage method [QAnywhere Java]" on page 525

browseMessagesByID method

Browse the message with the given message ID.

Syntax

```
java.util.Enumeration QAManagerBase.browseMessagesByID(
     String id
) throws QAException
```

Parameters

- **id** The message ID of the message.

Returns

An enumerator containing 0 or 1 messages.

Exceptions

- **QAException class** Thrown if there is a problem browsing the messages.

Remarks

The message is just being browsed, so it cannot be acknowledged. Use QAManagerBase.getMessage(String) to receive messages so that they can be acknowledged.

See also

- "QAManagerBase.browseMessagesByQueue method [QAnywhere Java]" on page 518
- "QAManagerBase.browseMessages method [QAnywhere Java]" on page 517
- "QAManagerBase.getMessage method [QAnywhere Java]" on page 525

browseMessagesByQueue method

Browses the available messages waiting that have been sent to the given address.

Syntax

```
java.util.Enumeration QAManagerBase.browseMessagesByQueue(
     String address
) throws QAException
```

Parameters

- **address** The address of the messages.

Returns

An enumerator over the available messages.

Exceptions

- **QAException class** Thrown if there is a problem browsing the messages.

Remarks

The messages are just being browsed, so they cannot be acknowledged.

Use the QAManagerBase.getMessage(String) method to receive messages so they can be acknowledged.

See also

- "QAManagerBase.browseMessagesByID method [QAnywhere Java]" on page 518
- "QAManagerBase.browseMessages method [QAnywhere Java]" on page 517
- "QAManagerBase.getMessage method [QAnywhere Java]" on page 525

browseMessagesBySelector method

Browse messages queued in the message store that satisfy the given selector.

Syntax

```
java.util.Enumeration QAManagerBase.browseMessagesBySelector(
     String selector
) throws QAException
```

Parameters

- **selector** The selector.

Returns

An enumerator over the available messages.

Exceptions

- **QAException class** Thrown if there is a problem browsing the messages.

Remarks

The message is just being browsed, so it cannot be acknowledged. Use QAManagerBase.getMessage(String) to receive messages so that they can be acknowledged.

See also

- "QAManagerBase.browseMessagesByQueue method [QAnywhere Java]" on page 518
- "QAManagerBase.browseMessages method [QAnywhere Java]" on page 517
- "QAManagerBase.browseMessagesByID method [QAnywhere Java]" on page 518
- "QAManagerBase.getMessage method [QAnywhere Java]" on page 525

cancelMessage method

Cancels the message with the given message ID.

Syntax

```
boolean QAManagerBase.cancelMessage(String id) throws QAException
```

Parameters

- **id** The message ID of the message to cancel.

Exceptions

- **QAException class** Thrown if there is a problem canceling the message.

Remarks

Puts a message into a canceled state before it is transmitted.

With the default delete rules of the QAnywhere Agent, canceled messages are eventually deleted from the message store.

Fails if the message is already in a final state, or if the message has been transmitted to the central messaging server.

close method

Closes the connection to the QAnywhere message system and releases any resources used by the QAManagerBase.

Syntax

```
void QAManagerBase.close() throws QAException
```

Exceptions

- **QAException class** Thrown if there is a problem closing the QAManagerBase instance.

Remarks

Additional calls to close() following the first are ignored. This method cannot be called in a message/exception listener.

If a database connection error is detected, you can re-open a QAManager by calling the close method followed by the open method. When re-opening a QAManager, you do not need to re-create it, reset the properties, or reset the message listeners. The properties of the QAManager cannot be changed after the first open, and subsequent open calls must supply the same acknowledgement mode.

createBinaryMessage method

Creates a QABinaryMessage object.

Syntax

```
QABinaryMessage QAManagerBase.createBinaryMessage() throws QAException
```

Returns

A new QABinaryMessage instance.

Exceptions

- **QAException class** Thrown if there is a problem creating the message.

Remarks

A QABinaryMessage object is used to send a message containing a message body of uninterpreted bytes.

See also

- "QABinaryMessage interface [QAnywhere Java]" on page 481

createTextMessage method

Creates a QATextMessage object.

Syntax

```
QATextMessage QAManagerBase.createTextMessage() throws QAException
```

Returns

A new QATextMessage instance.

Exceptions

- **QAException class** Thrown if there is a problem creating the message.

Remarks

A QATextMessage object is used to send a message containing a string message body.

See also

- "QATextMessage interface [QAnywhere Java]" on page 569

getBooleanStoreProperty method

Gets a boolean value for a predefined or custom message store property.

Syntax

```
boolean QAManagerBase.getBooleanStoreProperty(
     String name
) throws QAException
```

Parameters

- **name** The predefined or custom property name.

Returns

The boolean property value.

Exceptions

- **QAException class** Thrown if there is a conversion error or message store error getting the property value or if the property does not exist.

Remarks

You can use this method to access predefined or user-defined client store properties.

For a list of predefined properties, see MessageStoreProperties.

See also

- "MessageStoreProperties interface [QAnywhere Java]" on page 476

getByteStoreProperty method

Gets a signed byte value for a predefined or custom message store property.

Syntax

```
byte QAManagerBase.getByteStoreProperty(String name) throws QAException
```

Parameters

- **name** The predefined or custom property name.

Returns

The signed byte property value.

Exceptions

- **QAException class** Thrown if there is a conversion error or message store error getting the property value or if the property does not exist.

Remarks

You can use this method to access predefined or user-defined client store properties.

For a list of predefined properties, see MessageStoreProperties.

See also

- "MessageStoreProperties interface [QAnywhere Java]" on page 476

getDoubleStoreProperty method

Gets a double value for a predefined or custom message store property.

Syntax

```
double QAManagerBase.getDoubleStoreProperty(
     String name
) throws QAException
```

Parameters

- **name** the predefined or custom property name.

Returns

The double property value.

Exceptions

- **QAException class** Thrown if there is a conversion error or message store error getting the property value or if the property does not exist.

Remarks

You can use this method to access predefined or user-defined client store properties.

For a list of predefined properties, see MessageStoreProperties.

See also

- "MessageStoreProperties interface [QAnywhere Java]" on page 476

getFloatStoreProperty method

Gets a float value for a predefined or custom message store property.

Syntax

```
float QAManagerBase.getFloatStoreProperty(
     String name
) throws QAException
```

Parameters

- **name** The predefined or custom property name.

Returns

The float property value.

Exceptions

- **QAException class** Thrown if there is a conversion error or message store error getting the property value or if the property does not exist.

Remarks

You can use this method to access predefined or user-defined client store properties.

For a list of predefined properties, see MessageStoreProperties.

See also

- "MessageStoreProperties interface [QAnywhere Java]" on page 476

getIntStoreProperty method

Gets an int value for a predefined or custom message store property.

Syntax

```
int QAManagerBase.getIntStoreProperty(String name) throws QAException
```

Parameters

- **name** The predefined or custom property name.

Returns

The integer property value.

Exceptions

- **QAException class** Thrown if there is a conversion error or message store error getting the property value or if the property does not exist.

Remarks

You can use this method to access predefined or user-defined client store properties.

For a list of predefined properties, see MessageStoreProperties.

See also

- "MessageStoreProperties interface [QAnywhere Java]" on page 476

getLongStoreProperty method

Gets a long value for a predefined or custom message store property.

Syntax

```
long QAManagerBase.getLongStoreProperty(String name) throws QAException
```


Parameters

- **name** The predefined or custom property name.

Returns

The long property value.

Exceptions

- **QAException class** Thrown if there is a conversion error or message store error getting the property value or if the property does not exist.

Remarks

You can use this method to access predefined or user-defined client store properties.

For a list of predefined properties, see MessageStoreProperties.

See also

- "MessageStoreProperties interface [QAnywhere Java]" on page 476

getMessage method

Returns the next available QAMessage sent to the specified address.

Syntax

```
QAMessage QAManagerBase.getMessage(String address) throws QAException
```

Parameters

- **address** This address specifies the queue name used by the QAnywhere client to receive messages.

Returns

The next QAMessage, or null if no message is available.

Exceptions

- **QAException class** Thrown if there is a problem getting the message.

Remarks

The address parameter specifies a local queue name. The address can be in the form 'store-id\queue-name' or 'queue-name'. If there is no message available, this call blocks indefinitely until a message is available. Use this method to receive messages synchronously.

See also

- "QAMessage interface [QAnywhere Java]" on page 550

getMessageBySelector method

Returns the next available QAMessage sent to the specified address that satisfies the given selector.

Syntax

```
QAMessage QAManagerBase.getMessageBySelector(
     String address,
    String selector
) throws QAException
```

Parameters

- **address** This address specifies the queue name used by the QAnywhere client to receive messages.
- **selector** The selector.

Returns

The next QAMessage, or null if no message is available.

Exceptions

- **QAException class** Thrown if there is a problem getting the message.

Remarks

The address parameter specifies a local queue name. The address can be in the form 'store-id\queue-name' or 'queue-name'. If there is no message available, this call blocks indefinitely until a message is available.

Use this method to receive messages synchronously.

See also

- "QAMessage interface [QAnywhere Java]" on page 550

getMessageBySelectorNoWait method

Returns the next available QAMessage sent to the given address that satisfies the given selector.

Syntax

```
QAMessage QAManagerBase.getMessageBySelectorNoWait(
     String address,
    String selector
) throws QAException
```

Parameters

- **address** This address specifies the queue name used by the QAnywhere client to receive messages.
- **selector** The selector.

Returns

The next available QAMessage or null there is no available message.

Exceptions

- **QAException class** Thrown if there is a problem getting the message.

Remarks

The address parameter specifies a local queue name. The address can be in the form 'store-id\queue-name' or 'queue-name'. If no message is available, this method returns immediately.

Use this method to receive messages synchronously.

See also

- "QAMessage interface [QAnywhere Java]" on page 550

getMessageBySelectorTimeout method

Returns the next available QAMessage sent to the given address that satisfies the given selector.

Syntax

```
QAMessage QAManagerBase.getMessageBySelectorTimeout(
     String address,
    String selector,
    long timeout
) throws QAException
```

Parameters

- **address** This address specifies the queue name used by the QAnywhere client to receive messages.
- **selector** The selector.
- **timeout** The time to wait, in milliseconds, for a message to become available.

Returns

The next available QAMessage, or null if no message is available.

Exceptions

- **QAException class** Thrown if there is a problem getting the message.

Remarks

The address parameter specifies a local queue name. The address can be in the form 'store-id\queue-name' or 'queue-name'. If no message is available, this method waits for the specified timeout and then returns.

Use this method to receive messages synchronously.

See also

- "QAMessage interface [QAnywhere Java]" on page 550

getMessageNoWait method

Returns the next available QAMessage sent to the given address.

Syntax

```
QAMessage QAManagerBase.getMessageNoWait(
     String address
) throws QAException
```

Parameters

- **address** This address specifies the queue name used by the QAnywhere client to receive messages.

Returns

The next available QAMessage or null there is no available message.

Exceptions

- **QAException class** Thrown if there is a problem getting the message.

Remarks

The address parameter specifies a local queue name. The address can be in the form 'store-id\queue-name' or 'queue-name'. If no message is available, this method returns immediately.

Use this method to receive messages synchronously.

See also

- "QAMessage interface [QAnywhere Java]" on page 550

getMessageTimeout method

Returns the next available QAMessage sent to the given address.

Syntax

```
QAMessage QAManagerBase.getMessageTimeout(
     String address,
    long timeout
) throws QAException
```

Parameters

- **address** This address specifies the queue name used by the QAnywhere client to receive messages.
- **timeout** The time to wait, in milliseconds, for a message to become available.

Returns

The next QAMessage, or null if no message is available.

Exceptions

- **QAException class** Thrown if there is a problem getting the message.

Remarks

The address parameter specifies a local queue name. The address can be in the form 'store-id\queue-name' or 'queue-name'. If no message is available, this method waits for the specified timeout and then returns. Use this method to receive messages synchronously.

See also

- "QAMessage interface [QAnywhere Java]" on page 550

getMode method

Returns the QAManager acknowledgement mode for received messages.

Syntax

```
short QAManagerBase.getMode() throws QAException
```

Returns

The QAManager acknowledgement mode for received messages.

Exceptions

- **QAException class** Thrown if there is a problem retrieving the QAManager acknowledgement mode.

Remarks

For a list of return values, see AcknowledgementMode.

AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT and AcknowledgementMode.IMPLICIT_ACKNOWLEDGEMENT apply to QAManager instances. AcknowledgementMode.TRANSACTIONAL is the mode for QATransactionalManager instances.

See also

- "AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT variable [QAnywhere Java]" on page 469
- "AcknowledgementMode.IMPLICIT_ACKNOWLEDGEMENT variable [QAnywhere Java]" on page 469
- "QAManager interface [QAnywhere Java]" on page 507
- "QATransactionalManager interface [QAnywhere Java]" on page 575

getQueueDepth method

Returns the total depth of all queues, based on a given filter.

Overload list

Name	Description
getQueueDepth(short) method	Returns the total depth of all queues, based on a given filter.
getQueueDepth(String, short) method	Returns the depth of a queue, based on a given filter.

getQueueDepth(short) method

Returns the total depth of all queues, based on a given filter.

Syntax

```
int QAManagerBase.getQueueDepth(short filter) throws QAException
```

Parameters

- **filter** A filter indicating incoming messages, outgoing messages, or all messages.

Returns

The number of messages in all queues for the given filter.

Exceptions

- **QAException class** Thrown if there was an error.

Remarks

The incoming depth of the queue is the number of incoming messages that have not been received (for example, using the QAManagerBase.getMessage(String) method). The outgoing depth of a queue is the number of outgoing messages (including uncommitted) that have not been transmitted to the server.

For a list of possible filter values, see QueueDepthFilter.

See also

- "QAManagerBase.getMessage method [QAnywhere Java]" on page 525

getQueueDepth(String, short) method

Returns the depth of a queue, based on a given filter.

Syntax

```
int QAManagerBase.getQueueDepth(
     String queue,
```



```
    short filter
) throws QAException
```

Parameters

- **filter** A filter indicating incoming messages, outgoing messages, or all messages.
- **queue** The queue name.

Returns

The number of messages.

Exceptions

- **QAException class** Thrown if there was an error.

Remarks

The incoming depth of the queue is the number of incoming messages that have not been received (for example, using the QAManagerBase.getMessage(String) method). The outgoing depth of a queue is the number of outgoing messages (including uncommitted) that have not been transmitted to the server.

For a list of possible filter values, see QueueDepthFilter.

getShortStoreProperty method

Gets a short value for a predefined or custom message store property.

Syntax

```
short QAManagerBase.getShortStoreProperty(
     String name
) throws QAException
```

Parameters

- **name** The predefined or custom property name.

Returns

The short property value.

Exceptions

- **QAException class** Thrown if there is a conversion error or message store error getting the property value or if the property does not exist.

Remarks

You can use this method to access predefined or user-defined client store properties.

For a list of predefined properties, see MessageStoreProperties.

See also

- "MessageStoreProperties interface [QAnywhere Java]" on page 476

getStoreProperty method

Gets an Object representing a message store property.

Syntax

```
Object QAManagerBase.getStoreProperty(String name) throws QAException
```

Parameters

- **name** The predefined or custom property name.

Returns

The property value.

Exceptions

- **QAException class** Thrown if there is a problem retrieving the property.

Remarks

You can use this method to access predefined or user-defined client store properties.

For a list of predefined properties, see MessageStoreProperties.

See also

- "MessageStoreProperties interface [QAnywhere Java]" on page 476

getStorePropertyNames method

Gets an enumerator over the message store property names.

Syntax

```
java.util.Enumeration QAManagerBase.getStorePropertyNames()
    throws QAException
```

Returns

An enumerator over the message store property names.

Exceptions

- **QAException class** Thrown if there is a problem retrieving the enumerator.

getStringStoreProperty method

Gets a string value for a predefined or custom message store property.

Syntax

```
String QAManagerBase.getStringStoreProperty(
     String name
) throws QAException
```

Parameters

- **name** The predefined or custom property name.

Returns

The string property value or null if the property does not exist.

Exceptions

- **QAException class** Thrown if there is a problem retrieving the string value.

Remarks

You can use this method to access predefined or user-defined client store properties.

For a list of predefined properties, see MessageStoreProperties.

See also

- "MessageStoreProperties interface [QAnywhere Java]" on page 476

propertyExists method

Tests if there currently exists a value for the given the property.

Syntax

```
boolean QAManagerBase.propertyExists(String name) throws QAException
```

Parameters

- **name** The predefined or custom property name.

Returns

true if the message store has a value mapped to the property. false otherwise.

Exceptions

- **QAException class** Thrown if there is a problem retrieving the property value.

Remarks

You can use this method to determine if a given property name currently has a value mapped to it by the message store.

For a list of predefined properties, see MessageStoreProperties.

See also

- "MessageStoreProperties interface [QAnywhere Java]" on page 476

putMessage method

Prepares a message to send to another QAnywhere client.

Syntax

```
void QAManagerBase.putMessage(
     String address,
    QAMessage msg
) throws QAException
```

Parameters

- **address** The address of the message specifying the destination queue name.
- **msg** The message to put in the local message store for transmission.

Exceptions

- **QAException class** Thrown if there is a problem putting the message.

Remarks

This method inserts a message and a destination address into your local message store. The time of message transmission depends on QAnywhere Agent transmission policies.

The address takes the form 'id\queue-name', where 'id' is the destination message store id and 'queue-name' identifies a queue that is used by the destination QAnywhere client to listen for or receive messages.

See also

- "QAManagerBase.putMessageTimeToLive method [QAnywhere Java]" on page 534

putMessageTimeToLive method

Prepares a message to send to another QAnywhere client.

Syntax

```
void QAManagerBase.putMessageTimeToLive(
     String address,
```



```
    QAMessage msg,
    long ttl
) throws QAException
```

Parameters

- **address** The address of the message specifying the destination queue name.
- **msg** The message to put.
- **ttl** The delay, in milliseconds, before the message expires if it has not been delivered. A value of 0 indicates the message does not expire.

Exceptions

- **QAException class** Thrown if there is a problem putting the message.

Remarks

This method inserts a message and a destination address into your local message store. The time of message transmission depends on QAnywhere Agent transmission policies. However, if the next message transmission time exceeds the given time-to-live value, the message expires.

The address takes the form 'id\queue-name', where 'id' is the destination message store id and 'queue-name' identifies a queue that is used by the destination QAnywhere client to listen for or receive messages.

reOpen method

Reopens the QAManagerBase.

Syntax

```
void QAManagerBase.reOpen() throws QAException
```

Exceptions

- **QAException class** Thrown if there is a problem reopening the QAManagerBase instance.

Remarks

This re-establishes connections to the message store, without releasing any resources. This method may called in a message or exception listener, and in that case it is not necessary to call start() again. This method simply executes close() then open() if not called in a listener, and in that case start() must be called to restart receiving of messages.

setBooleanStoreProperty method

Sets a predefined or custom message store property to a boolean value.

Syntax

```
void QAManagerBase.setBooleanStoreProperty(
     String name,
    boolean value
) throws QAException
```

Parameters

- **name** The predefined or custom property name.
- **value** The boolean property value.

Exceptions

- **QAException class** Thrown if there is a problem setting the message store property.

Remarks

You can use this method to set predefined or user-defined client store properties.

For a list of predefined properties, see MessageStoreProperties.

See also

- "MessageStoreProperties interface [QAnywhere Java]" on page 476

setByteStoreProperty method

Sets a predefined or custom message store property to a sbyte value.

Syntax

```
void QAManagerBase.setByteStoreProperty(
     String name,
    byte value
) throws QAException
```

Parameters

- **name** The predefined or custom property name.
- **value** The sbyte property value.

Exceptions

- **QAException class** Thrown if there is a problem setting the message store property.

Remarks

You can use this method to set predefined or user-defined client store properties.

For a list of predefined properties, see MessageStoreProperties.

See also

- "MessageStoreProperties interface [QAnywhere Java]" on page 476

setDoubleStoreProperty method

Sets a predefined or custom message store property to a double value.

Syntax

```
void QAManagerBase.setDoubleStoreProperty(
     String name,
    double value
) throws QAException
```

Parameters

- **name** The predefined or custom property name.
- **value** The double property value.

Exceptions

- **QAException class** Thrown if there is a problem setting the message store property.

Remarks

You can use this method to set predefined or user-defined client. store properties.

For a list of predefined properties, see MessageStoreProperties.

See also

- "MessageStoreProperties interface [QAnywhere Java]" on page 476

setFloatStoreProperty method

Sets a predefined or custom message store property to a float value.

Syntax

```
void QAManagerBase.setFloatStoreProperty(
     String name,
    float value
) throws QAException
```

Parameters

- **name** The predefined or custom property name.
- **value** The float property value.

Exceptions

- **QAException class** Thrown if there is a problem setting the message store property.

Remarks

You can use this method to set predefined or user-defined client store properties. For a list of predefined properties, see MessageStoreProperties.

See also

- "MessageStoreProperties interface [QAnywhere Java]" on page 476

setIntStoreProperty method

Sets a predefined or custom message store property to an int value.

Syntax

```
void QAManagerBase.setIntStoreProperty(
     String name,
    int value
) throws QAException
```

Parameters

- **name** The predefined or custom property name.
- **value** The int property value.

Exceptions

- **QAException class** Thrown if there is a problem setting the message store property.

Remarks

You can use this method to set predefined or user-defined client store properties. For a list of predefined properties, see MessageStoreProperties.

See also

- "MessageStoreProperties interface [QAnywhere Java]" on page 476

setLongStoreProperty method

Sets a predefined or custom message store property to a long value.

Syntax

```
void QAManagerBase.setLongStoreProperty(
     String name,
    long value
) throws QAException
```


Parameters

- **name** The predefined or custom property name.
- **value** The long property value.

Exceptions

- **QAException class** Thrown if there is a problem setting the message store property.

Remarks

You can use this method to set predefined or user-defined client store properties.

For a list of predefined properties, see MessageStoreProperties.

See also

- "MessageStoreProperties interface [QAnywhere Java]" on page 476

setMessageListener method

Registers a QAMessageListener object to receive QAnywhere messages asynchronously.

Syntax

```
void QAManagerBase.setMessageListener(
     String address,
    QAMessageListener listener
) throws QAException
```

Parameters

- **address** The address of a local queue name used to receive messages, or system to listen for QAnywhere system messages.
- **listener** The listener.

Exceptions

- **QAException class** Thrown if there is a problem registering the QAMessageListener object.

Remarks

The address parameter specifies a local queue name used to receive the message. You can only have one listener object assigned to a given queue. If you want to listen for QAnywhere system messages, including push notifications and network status changes, specify "system" as the queue name.

Use this method to receive messages asynchronously.

See also

- "QAMessageListener interface [QAnywhere Java]" on page 566

setMessageListener2 method

Registers a QAMessageListener2 object to receive QAnywhere messages asynchronously.

Syntax

```
void QAManagerBase.setMessageListener2(
     String address,
    QAMessageListener2 listener
) throws QAException
```

Parameters

- **address** The address of a local queue name used to receive messages, or system to listen for QAnywhere system messages.
- **listener** The listener.

Exceptions

- **QAException class** Thrown if there is a problem registering the QAMessageListener2 object.

Remarks

The address parameter specifies a local queue name used to receive the message. You can only have one listener object assigned to a given queue. If you want to listen for QAnywhere system messages, including push notifications and network status changes, specify "system" as the queue name.

Use this method to receive messages asynchronously.

See also

- "QAMessageListener2 interface [QAnywhere Java]" on page 568

setMessageListenerBySelector method

Registers a QAMessageListener object to receive QAnywhere messages asynchronously, with a message selector.

Syntax

```
void QAManagerBase.setMessageListenerBySelector(
     String address,
    String selector,
    QAMessageListener listener
) throws QAException
```

Parameters

- **address** The address of a local queue name used to receive messages, or system to listen for QAnywhere system messages.
- **selector** The selector to be used to filter the messages to be received.

- **listener** The listener.

Exceptions

- **QAException class** Thrown if there is a problem registering the QAMessageListener object, such as because there is already a listener object assigned to the given queue.

Remarks

The address parameter specifies a local queue name used to receive the message. You can only have one listener object assigned to a given queue. The selector parameter specifies a selector to be used to filter the messages to be received on the given address. If you want to listen for QAnywhere system messages, including push notifications and network status changes, specify "system" as the queue name.

Use this method to receive messages asynchronously.

setMessageListenerBySelector2 method

Registers a QAMessageListener2 object to receive QAnywhere messages asynchronously, with a message selector.

Syntax

```
void QAManagerBase.setMessageListenerBySelector2(
     String address,
    String selector,
    QAMessageListener2 listener
) throws QAException
```

Parameters

- **address** The address of a local queue name used to receive messages, or system to listen for QAnywhere system messages.
- **selector** The selector to be used to filter the messages to be received.
- **listener** The listener.

Exceptions

- **QAException class** Thrown if there is a problem registering the QAMessageListener2 object.

Remarks

The address parameter specifies a local queue name used to receive the message. You can only have one listener object assigned to a given queue. The selector parameter specifies a selector to be used to filter the messages to be received on the given address. If you want to listen for QAnywhere system messages, including push notifications and network status changes, specify "system" as the queue name.

Use this method to receive messages asynchronously.

See also

- "QAMessageListener2 interface [QAnywhere Java]" on page 568

setProperty method

Allows you to set QAnywhere Manager configuration properties programmatically.

Syntax

```
void QAManagerBase.setProperty(
     String name,
    String val
) throws QAException
```

Parameters

- **name** The QAnywhere Manager configuration property name.
- **val** The QAnywhere Manager configuration property value.

Exceptions

- **QAException class** Thrown if there is a problem setting the property to the value.

Remarks

You can use this method to override default QAnywhere Manager configuration properties by specifying a property name and value. For a list of properties, see MessageStoreProperties.

You can also set QAnywhere Manager configuration properties using a properties file and the QAManagerFactory.CreateQAManager method. For more information, see Setting properties using a properties files and the QAManagerFactory.CreateQAManager method.

Note
You must set required properties before calling QAManager.Open or QATransactionalManager.Open().

See also

- "MessageStoreProperties interface [QAnywhere Java]" on page 476

setShortStoreProperty method

Sets a predefined or custom message store property to a short value.

Syntax

```
void QAManagerBase.setShortStoreProperty(
     String name,
    short value
) throws QAException
```

Parameters

- **name** The predefined or custom property name.
- **value** The short property value.

Exceptions

- **QAException class** Thrown if there is a problem setting the message store property.

Remarks

You can use this method to set predefined or user-defined client store properties.

For a list of predefined properties, see MessageStoreProperties.

See also

- "MessageStoreProperties interface [QAnywhere Java]" on page 476

setStoreProperty method

Sets a predefined or custom message store property to a System.Object value.

Syntax

```
void QAManagerBase.setStoreProperty(
     String name,
    Object value
) throws QAException
```

Parameters

- **name** The predefined or custom property name.
- **value** The property value.

Exceptions

- **QAException class** Thrown if there is a problem setting the message store property to the value.

Remarks

The property type must correspond to one of the acceptable primitive types, or String. You can use this method to set predefined or user-defined client store properties.

For a list of predefined properties, see MessageStoreProperties.

See also

- "MessageStoreProperties interface [QAnywhere Java]" on page 476

setStringStoreProperty method

Sets a predefined or custom message store property to a String value.

Syntax

```
void QAManagerBase.setStringStoreProperty(
     String name,
```

```
    String value
) throws QAException
```

Parameters

- **name** The predefined or custom property name.
- **value** The String property value.

Exceptions

- **QAException class** Thrown if there is a problem setting the message store property to a string value.

Remarks

You can use this method to set predefined or user-defined client store properties.

For a list of predefined properties, see MessageStoreProperties.

See also

- "MessageStoreProperties interface [QAnywhere Java]" on page 476

start method

Starts the QAManagerBase for receiving incoming messages.

Syntax

```
void QAManagerBase.start() throws QAException
```

Exceptions

- **QAException class** Thrown if there is a problem starting the QAManagerBase instance.

Remarks

Any calls to this method beyond the first without an intervening QAManagerBase.stop() call are ignored.

See also

- "QAManagerBase.stop method [QAnywhere Java]" on page 544

stop method

Halts the QAManagerBase's reception of incoming messages.

Syntax

```
void QAManagerBase.stop() throws QAException
```


Exceptions

- **QAException class** Thrown if there is a problem stopping the QAManagerBase instance.

Remarks

The messages are not lost. They just are not received until the manager is started again. Any calls to stop() beyond the first without an intervening QAManagerBase.start() call are ignored.

See also

- "QAManagerBase.start method [QAnywhere Java]" on page 544

triggerSendReceive method

Causes a synchronization with the QAnywhere message server, uploading any messages addressed to other clients, and downloading any messages addressed to the local client.

Syntax

```
void QAManagerBase.triggerSendReceive() throws QAException
```

Exceptions

- **QAException class** Thrown if there is a problem triggering the send/receive.

Remarks

A call to this method results in immediate message synchronization between a QAnywhere Agent and the central messaging server. A manual triggerSendReceive() call results in immediate message transmission, independent of the QAnywhere Agent transmission policies.

QAnywhere Agent transmission policies determine how message transmission occurs. For example, message transmission can occur automatically at regular intervals, when your client receives a push notification, or when you call the QAManagerBase.putMessage() method to send a message.

See also

- "QAManagerBase.putMessage method [QAnywhere Java]" on page 534

QAManagerFactory class

This class acts as a factory class for creating QATransactionalManager and QAManager objects.

Syntax

```
public class QAManagerFactory
```

Members

All members of QAManagerFactory class, including all inherited members.

Name	Description
createQAManager method	Returns a new QAManager instance with the specified properties.
createQATransactionalManager method	Returns a new QATransactionalManager instance with the specified properties.
getInstance method	Returns the singleton QAManagerFactory instance.

Remarks

You can only have one instance of QAManagerFactory.

See also

- "QAManager interface [QAnywhere Java]" on page 507
- "QATransactionalManager interface [QAnywhere Java]" on page 575

createQAManager method

Returns a new QAManager instance with the specified properties.

Overload list

Name	Description
createQAManager() method	Returns a new QAManager instance with default properties.
createQAManager(Hashtable) method	Returns a new QAManager instance with the specified properties as a Hashtable.
createQAManager(String) method	Returns a new QAManager instance with the specified properties.

createQAManager() method

Returns a new QAManager instance with default properties.

Syntax

```
abstract QAManager QAManagerFactory.createQAManager() throws QAException
```

Returns

A new QAManager instance.

Exceptions

- **QAException class** Thrown if there is a problem creating the manager.

See also

- "QAManager interface [QAnywhere Java]" on page 507

createQAManager(Hashtable) method

Returns a new QAManager instance with the specified properties as a Hashtable.

Syntax

```
abstract QAManager QAManagerFactory.createQAManager(
     java.util.Hashtable properties
) throws QAException
```

Parameters

- **properties** A Hashtable for configuring the QAManager instance.

Returns

A new QAManager instance.

Exceptions

- **QAException class** Thrown if there is a problem creating the manager.

See also

- "QAManager interface [QAnywhere Java]" on page 507

createQAManager(String) method

Returns a new QAManager instance with the specified properties.

Syntax

```
abstract QAManager QAManagerFactory.createQAManager(
     String iniFile
) throws QAException
```

Parameters

- **iniFile** A properties file for configuring the QAManager instance, or null to create the QAManager instance using default properties.

Returns

A new QAManager instance.

Exceptions

- **QAException class** Thrown if there is a problem creating the manager.

Remarks

If the iniFile parameter is null, the QAManager is created using default properties. You can use the QAManagerBase set property methods to set QAManager properties programmatically after you create the instance.

See also

- "QAManager interface [QAnywhere Java]" on page 507

createQATransactionalManager method

Returns a new QATransactionalManager instance with the specified properties.

Overload list

Name	Description
createQATransactionalManager() method	Returns a new QATransactionalManager instance with default properties.
createQATransactionalManager(Hashtable) method	Returns a new QATransactionalManager instance with the specified properties.
createQATransactionalManager(String) method	Returns a new QATransactionalManager instance with the specified properties.

createQATransactionalManager() method

Returns a new QATransactionalManager instance with default properties.

Syntax

```
abstract QATransactionalManager
QAManagerFactory.createQATransactionalManager()
    throws QAException
```

Returns

A new QATransactionalManager.

Exceptions

- **QAException class** Thrown if there is a problem creating the manager.

See also

- "QATransactionalManager interface [QAnywhere Java]" on page 575

createQATransactionalManager(Hashtable) method

Returns a new QATransactionalManager instance with the specified properties.

Syntax

```
abstract QATransactionalManager
QAManagerFactory.createQATransactionalManager(
     java.util.Hashtable properties
) throws QAException
```

Parameters

- **properties** A hashtable for configuring the QATransactionalManager instance.

Returns

The configured QATransactionalManager.

Exceptions

- **QAException class** Thrown if there is a problem creating the manager.

See also

- "QATransactionalManager interface [QAnywhere Java]" on page 575

createQATransactionalManager(String) method

Returns a new QATransactionalManager instance with the specified properties.

Syntax

```
abstract QATransactionalManager
QAManagerFactory.createQATransactionalManager(
     String iniFile
) throws QAException
```

Parameters

- **iniFile** A properties file for configuring the QATransactionalManager instance.

Returns

The configured QATransactionalManager.

Exceptions

- **QAException class** Thrown if there is a problem creating the manager.

Remarks

If the iniFile parameter is null, the QATransactionalManager is created using default properties. You can use the QAManagerBase set property methods to set QATransactionalManager properties programmatically after you create the instance.

See also

- "QATransactionalManager interface [QAnywhere Java]" on page 575

getInstance method

Returns the singleton QAManagerFactory instance.

Syntax

```
QAManagerFactory QAManagerFactory.getInstance() throws QAException
```

Returns

The singleton QAManagerFactory instance.

Exceptions

- **QAException class** Thrown if there is a problem creating the manager factory.

QAMessage interface

QAMessage provides an interface to set message properties and header fields.

Syntax

```
public interface QAMessage
```

Derived classes

- "QABinaryMessage interface [QAnywhere Java]" on page 481
- "QATextMessage interface [QAnywhere Java]" on page 569

Members

All members of QAMessage interface, including all inherited members.

Name	Description
clearProperties method	Clear all the properties of the message.
getAddress method	Returns the destination address for the QAMessage instance.
getBooleanProperty method	Gets a boolean message property.
getByteProperty method	Gets a signed byte message property.
getDoubleProperty method	Gets a double message property.

Name	Description
getExpiration method	Returns the message's expiration value, or null if the message does not expire or has not yet been sent.
getFloatProperty method	Gets a float message property.
getInReplyToID method	Returns the message ID of the message to which this message is a reply.
getIntProperty method	Gets an int message property.
getLongProperty method	Gets a long message property.
getMessageID method	Returns the globally unique message ID of the message.
getPriority method	Returns the priority of the message (ranging from 0 to 9).
getProperty method	Gets a message property.
getPropertyNames method	Gets an enumerator over the property names of the message.
getPropertyType method	Returns the property type of the given property.
getRedelivered method	Indicates whether the message has been previously received but not acknowledged.
getReplyToAddress method	Returns the reply-to address of this message.
getShortProperty method	Gets a short message property.
getStringProperty method	Gets a String message property.
getTimestamp method	Returns the message timestamp, which is the time the message was created.
propertyExists method	Indicates whether the given property has been set for this message.
setBooleanProperty method	Sets a boolean property.
setByteProperty method	Sets a signed byte property.
setDoubleProperty method	Sets a double property.
setFloatProperty method	Sets a float property.
setInReplyToID method	Sets the in reply to ID, which identifies the message this message is a reply to.

Name	Description
setIntProperty method	Sets an int property.
setLongProperty method	Sets a long property.
setPriority method	Sets the priority of the message (ranging from 0 to 9).
setProperty method	Sets a property.
setReplyToAddress method	Sets the reply-to address.
setShortProperty method	Sets a short property.
setStringProperty method	Sets a string property.
DEFAULT_PRIORITY variable	The default message priority.
DEFAULT_TIME_TO_LIVE variable	The default time-to-live value.

Remarks

The derived classes QABinaryMessage and QATextMessage provide specialized functions to read and write to the message body. You can use QAMessage functions to set predefined or custom message properties.

For a list of predefined property names, see the MessageProperties.

See also

- "QABinaryMessage interface [QAnywhere Java]" on page 481
- "QATextMessage interface [QAnywhere Java]" on page 569

clearProperties method

Clear all the properties of the message.

Syntax

```
void QAMessage.clearProperties() throws QAException
```

Exceptions

- **QAException class** Thrown if there is a problem clearing the message properties.

getAddress method

Returns the destination address for the QAMessage instance.

Syntax

```
String QAMessage.getAddress() throws QAException
```

Returns

The destination address for the QAMessage instance.

Exceptions

- **QAException class** Thrown if there is a problem retrieving the destination address.

Remarks

When a message is sent, this field is ignored. After completion of a send operation, the field holds the destination address specified in QAManagerBase.putMessage(String, QAMessage).

getBooleanProperty method

Gets a boolean message property.

Syntax

```
boolean QAMessage.getBooleanProperty(String name) throws QAException
```

Parameters

- **name** The property name.

Returns

The property value.

Exceptions

- **QAException class** Thrown if there is a conversion error getting the property value or if the property does not exist.

See also

- "MessageProperties interface [QAnywhere Java]" on page 469

getByteProperty method

Gets a signed byte message property.

Syntax

```
byte QAMessage.getByteProperty(String name) throws QAException
```

Parameters

- **name** The property name.

Returns

The property value.

Exceptions

- **QAException class** Thrown if there is a conversion error getting the property value or if the property does not exist.

See also

- "MessageProperties interface [QAnywhere Java]" on page 469

getDoubleProperty method

Gets a double message property.

Syntax

```
double QAMessage.getDoubleProperty(String name) throws QAException
```

Parameters

- **name** The property name.

Returns

The property value.

Exceptions

- **QAException class** Thrown if there is a conversion error getting the property value or if the property does not exist.

See also

- "MessageProperties interface [QAnywhere Java]" on page 469

getExpiration method

Returns the message's expiration value, or null if the message does not expire or has not yet been sent.

Syntax

```
java.util.Date QAMessage.getExpiration() throws QAException
```

Returns

The message's expiration value, or null if the message does not expire or has not yet been sent.

Exceptions

- **QAException class** Thrown if there is a problem getting the expiration.

Remarks

When a message is sent, the expiration is left unassigned. After the send operation completes, it holds the expiration time of the message.

This is a read-only property because the expiration time of a message is set by adding the time-to-live argument of QAManagerBase.putMessageTimeToLive(String, QAMessage, long) to the current time.

See also

- "QAManagerBase.putMessageTimeToLive method [QAnywhere Java]" on page 534

getFloatProperty method

Gets a float message property.

Syntax

```
float QAMessage.getFloatProperty(String name) throws QAException
```

Parameters

- **name** The property name.

Returns

The property value.

Exceptions

- **QAException class** Thrown if there is a conversion error getting the property value or if the property does not exist.

See also

- "MessageProperties interface [QAnywhere Java]" on page 469

getInReplyToID method

Returns the message ID of the message to which this message is a reply.

Syntax

```
String QAMessage.getInReplyToID() throws QAException
```

Returns

The message ID of the message to which this message is a reply, or null if this message is not a reply.

Exceptions

- **QAException class** Thrown if there is a problem getting the message ID of the message to which this message is a reply.

getIntProperty method

Gets an int message property.

Syntax

```
int QAMessage.getIntProperty(String name) throws QAException
```

Parameters

- **name** The property name.

Returns

The property value.

Exceptions

- **QAException class** Thrown if there is a conversion error getting the property value or if the property does not exist.

See also

- "MessageProperties interface [QAnywhere Java]" on page 469

getLongProperty method

Gets a long message property.

Syntax

```
long QAMessage.getLongProperty(String name) throws QAException
```

Parameters

- **name** The property name.

Returns

The property value.

Exceptions

- **QAException class** Thrown if there is a conversion error getting the property value or if the property does not exist.

See also

- "MessageProperties interface [QAnywhere Java]" on page 469

getMessageID method

Returns the globally unique message ID of the message.

Syntax

```
String QAMessage.getMessageID() throws QAException
```

Returns

The message ID of the message, or null if the message has not yet been put.

Exceptions

- **QAException class** Thrown if there is a problem getting the message ID.

Remarks

This property is null until a message is put.

When a message is sent using QAManagerBase.putMessage(String, QAMessage) the message ID is null and can be ignored. When the send method returns, it contains an assigned value.

See also

- "QAManagerBase.putMessage method [QAnywhere Java]" on page 534

getPriority method

Returns the priority of the message (ranging from 0 to 9).

Syntax

```
int QAMessage.getPriority() throws QAException
```

Returns

The priority of the message.

Exceptions

- **QAException class** Thrown if there is a problem getting the message priority.

getProperty method

Gets a message property.

Syntax

```
Object QAMessage.getProperty(String name) throws QAException
```

Parameters

- **name** The property name.

Returns

The property value, or null if the property does not exist.

Exceptions

- **QAException class** Thrown if there is a conversion error getting the property value.

getPropertyNames method

Gets an enumerator over the property names of the message.

Syntax

```
java.util.Enumeration QAMessage.getPropertyNames() throws QAException
```

Returns

An enumerator over the message property names.

Exceptions

- **QAException class** Thrown if there is a problem getting the enumerator over the property names of the message.

getPropertyType method

Returns the property type of the given property.

Syntax

```
short QAMessage.getPropertyType(String name) throws QAException
```

Parameters

- **name** The property name.

Returns

The property type.

Exceptions

- **QAException class** Thrown if there is a problem retrieving the property type.

See also

- "PropertyType interface [QAnywhere Java]" on page 479

getRedelivered method

Indicates whether the message has been previously received but not acknowledged.

Syntax

```
boolean QAMessage.getRedelivered() throws QAException
```

Returns

True if the message has been previously received but not acknowledged.

Exceptions

- **QAException class** Thrown if there is a problem retrieving the redelivered status.

Remarks

Redelivered is set by a receiving QAManager when it detects that a message being received was received before.

For example, an application receives a message using a QAManager opened with AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT and shuts down without acknowledging the message. When the application starts again and receives the same message, the message will be marked as redelivered.

getReplyToAddress method

Returns the reply-to address of this message.

Syntax

```
String QAMessage.getReplyToAddress() throws QAException
```

Returns

The reply-to address of this message, or null if it does not exist.

Exceptions

- **QAException class** Thrown if there is a problem retrieving the reply-to address.

getShortProperty method

Gets a short message property.

Syntax

```
short QAMessage.getShortProperty(String name) throws QAException
```

Parameters

- **name** the property name.

Returns

The property value.

Exceptions

- **QAException class** Thrown if there is a conversion error getting the property value or if the property does not exist.

See also

- "MessageProperties interface [QAnywhere Java]" on page 469

getStringProperty method

Gets a String message property.

Syntax

```
String QAMessage.getStringProperty(String name) throws QAException
```

Parameters

- **name** The property name.

Returns

The property value, or null if the property does not exist.

Exceptions

- **QAException class** Thrown if there is a problem retrieving the message property.

See also

- "MessageProperties interface [QAnywhere Java]" on page 469

getTimestamp method

Returns the message timestamp, which is the time the message was created.

Syntax

```
java.util.Date QAMessage.getTimestamp() throws QAException
```

Returns

The message timestamp.

Exceptions

- **QAException class** Thrown if there is a problem retrieving the message timestamp.

propertyExists method

Indicates whether the given property has been set for this message.

Syntax

```
boolean QAMessage.propertyExists(String name) throws QAException
```

Parameters

- **name** The property name

Returns

True if the property exists.

Exceptions

- **QAException class** Thrown if there is a problem checking if the property has been set.

setBooleanProperty method

Sets a boolean property.

Syntax

```
void QAMessage.setBooleanProperty(
     String name,
    boolean value
) throws QAException
```

Parameters

- **name** The property name.
- **value** The property value.

Exceptions

- **QAException class** Thrown if there is a problem setting the property.

See also

- "MessageProperties interface [QAnywhere Java]" on page 469

setByteProperty method

Sets a signed byte property.

Syntax

```
void QAMessage.setByteProperty(
     String name,
    byte value
) throws QAException
```

Parameters

- **name** The property name.
- **value** The property value.

Exceptions

- **QAException class** Thrown if there is a problem setting the property.

See also

- "MessageProperties interface [QAnywhere Java]" on page 469

setDoubleProperty method

Sets a double property.

Syntax

```
void QAMessage.setDoubleProperty(
     String name,
    double value
) throws QAException
```

Parameters

- **name** The property name.
- **value** The property value.

Exceptions

- **QAException class** Thrown if there is a problem setting the property.

See also

- "MessageProperties interface [QAnywhere Java]" on page 469

setFloatProperty method

Sets a float property.

Syntax

```
void QAMessage.setFloatProperty(
     String name,
    float value
) throws QAException
```

Parameters

- **name** The property name.
- **value** The property value.

Exceptions

- **QAException class** Thrown if there is a problem setting the property.

See also

- "MessageProperties interface [QAnywhere Java]" on page 469

setInReplyToID method

Sets the in reply to ID, which identifies the message this message is a reply to.

Syntax

```
void QAMessage.setInReplyToID(String id) throws QAException
```

Parameters

- **id** The ID of the message this message is in reply to.

Exceptions

- **QAException class** Thrown if there is a problem setting the in reply to ID.

setIntProperty method

Sets an int property.

Syntax

```
void QAMessage.setIntProperty(String name, int value) throws QAException
```

Parameters

- **name** The property name.
- **value** The property value.

Exceptions

- **QAException class** Thrown if there is a problem setting the property.

See also

- "MessageProperties interface [QAnywhere Java]" on page 469

setLongProperty method

Sets a long property.

Syntax

```
void QAMessage.setLongProperty(
     String name,
    long value
) throws QAException
```

Parameters

- **name** The property name.
- **value** The property value.

Exceptions

- **QAException class** Thrown if there is a problem setting the property.

See also

- "MessageProperties interface [QAnywhere Java]" on page 469

setPriority method

Sets the priority of the message (ranging from 0 to 9).

Syntax

```
void QAMessage.setPriority(int priority) throws QAException
```

Parameters

- **priority** The priority of the message.

Exceptions

- **QAException class** Thrown if there is a problem setting the priority.

setProperty method

Sets a property.

Syntax

```
void QAMessage.setProperty(String name, Object value) throws QAException
```


Parameters

- **name** The property name.
- **value** The property value.

Exceptions

- **QAException class** Thrown if there is a problem setting the property.

Remarks

The property type must correspond to one of the acceptable primitive types, or String.

See also

- "MessageProperties interface [QAnywhere Java]" on page 469

setReplyToAddress method

Sets the reply-to address.

Syntax

```
void QAMessage.setReplyToAddress(String address) throws QAException
```

Parameters

- **address** The reply-to address.

Exceptions

- **QAException class** Thrown if there is a problem setting the reply-to address.

setShortProperty method

Sets a short property.

Syntax

```
void QAMessage.setShortProperty(
     String name,
    short value
) throws QAException
```

Parameters

- **name** The property name.
- **value** The property value.

Exceptions

- **QAException class** Thrown if there is a problem setting the property.

See also

- "MessageProperties interface [QAnywhere Java]" on page 469

setStringProperty method

Sets a string property.

Syntax

```
void QAMessage.setStringProperty(
     String name,
    String value
) throws QAException
```

Parameters

- **name** The property name.
- **value** The property value.

Exceptions

- **QAException class** Thrown if there is a problem setting the property.

See also

- "MessageProperties interface [QAnywhere Java]" on page 469

DEFAULT_PRIORITY variable

The default message priority.

Syntax

```
final int QAMessage.DEFAULT_PRIORITY
```

DEFAULT_TIME_TO_LIVE variable

The default time-to-live value.

Syntax

```
final long QAMessage.DEFAULT_TIME_TO_LIVE
```

QAMessageListener interface

To listen for messages, implement this interface and register your implementation by calling QAMangerBase.setMessageListener(String,QAMessageListener).

Syntax

```
public interface QAMessageListener
```

Members

All members of QAMessageListener interface, including all inherited members.

Name	Description
onException method	This method is called whenever an exception occurs while listening for messages.
onMessage method	This method is called whenever a message is received.

See also

- "QAManagerBase.setMessageListener method [QAnywhere Java]" on page 539

onException method

This method is called whenever an exception occurs while listening for messages.

Syntax

```
void QAMessageListener.onException(
     QAException exception,
    QAMessage message
)
```

Parameters

- **exception** The exception that occurred.
- **message** If the exception occurred after the message was passed to onMessage(QAMessage), the message that was processed. Otherwise, null.

Remarks

Note that this method cannot be used to automatically close the QAManagerBase instance, as the QAManagerBase.close() method blocks until all message listeners are finished processing.

See also

- "QAManagerBase interface [QAnywhere Java]" on page 514
- "QAManagerBase.close method [QAnywhere Java]" on page 520

onMessage method

This method is called whenever a message is received.

Syntax

```
void QAMessageListener.onMessage(QAMessage message) throws QAException
```

Parameters

- **message** The message that was received.

QAMessageListener2 interface

To listen for messages, implement this interface and register your implementation by calling QAMangerBase.setMessageListener2(String,QAMessageListener2).

Syntax

```
public interface QAMessageListener2
```

Members

All members of QAMessageListener2 interface, including all inherited members.

Name	Description
onException method	This method is called whenever an exception occurs while listening for messages.
onMessage method	This method is called whenever a message is received.

See also

- "QAManagerBase.setMessageListener2 method [QAnywhere Java]" on page 540

onException method

This method is called whenever an exception occurs while listening for messages.

Syntax

```
void QAMessageListener2.onException(
     QAManagerBase mgr,
    QAException exception,
    QAMessage message
)
```

Parameters

- **mgr** The QAManagerBase that processed the message.
- **exception** The exception that occurred.
- **message** If the exception occurred after the message was passed to onMessage(QAMessage), the message that was processed. Otherwise, null.

Remarks

Note that this method cannot be used to automatically close the QAManagerBase instance, as the QAManagerBase.close() method blocks until all message listeners are finished processing.

See also

- "QAManagerBase interface [QAnywhere Java]" on page 514
- "QAManagerBase.close method [QAnywhere Java]" on page 520

onMessage method

This method is called whenever a message is received.

Syntax

```
void QAMessageListener2.onMessage(
     QAManagerBase mgr,
    QAMessage message
) throws QAException
```

Parameters

- **mgr** The QAManagerBase that received the message.
- **message** The message that was received.

See also

- "QAManagerBase interface [QAnywhere Java]" on page 514

QATextMessage interface

QATextMessage inherits from the QAMessage class and adds a text message body, and methods to read from and write to the text message body.

Syntax

```
public interface QATextMessage
```

Base classes

- "QAMessage interface [QAnywhere Java]" on page 550

Members

All members of QATextMessage interface, including all inherited members.

Name	Description
clearProperties method	Clear all the properties of the message.

Name	Description
getAddress method	Returns the destination address for the QAMessage instance.
getBooleanProperty method	Gets a boolean message property.
getByteProperty method	Gets a signed byte message property.
getDoubleProperty method	Gets a double message property.
getExpiration method	Returns the message's expiration value, or null if the message does not expire or has not yet been sent.
getFloatProperty method	Gets a float message property.
getInReplyToID method	Returns the message ID of the message to which this message is a reply.
getIntProperty method	Gets an int message property.
getLongProperty method	Gets a long message property.
getMessageID method	Returns the globally unique message ID of the message.
getPriority method	Returns the priority of the message (ranging from 0 to 9).
getProperty method	Gets a message property.
getPropertyNames method	Gets an enumerator over the property names of the message.
getPropertyType method	Returns the property type of the given property.
getRedelivered method	Indicates whether the message has been previously received but not acknowledged.
getReplyToAddress method	Returns the reply-to address of this message.
getShortProperty method	Gets a short message property.
getStringProperty method	Gets a String message property.
getText method	Returns the message text.
getTextLength method	Returns the length, in characters, of the message.
getTimestamp method	Returns the message timestamp, which is the time the message was created.

Name	Description
propertyExists method	Indicates whether the given property has been set for this message.
readText method	Returns unread text from the message.
reset method	Resets the text position of the message to the beginning.
setBooleanProperty method	Sets a boolean property.
setByteProperty method	Sets a signed byte property.
setDoubleProperty method	Sets a double property.
setFloatProperty method	Sets a float property.
setInReplyToID method	Sets the in reply to ID, which identifies the message this message is a reply to.
setIntProperty method	Sets an int property.
setLongProperty method	Sets a long property.
setPriority method	Sets the priority of the message (ranging from 0 to 9).
setProperty method	Sets a property.
setReplyToAddress method	Sets the reply-to address.
setShortProperty method	Sets a short property.
setStringProperty method	Sets a string property.
setText method	Overwrites the message text.
writeText method	Appends text to the text of the message.
DEFAULT_PRIORITY variable	The default message priority.
DEFAULT_TIME_TO_LIVE variable	The default time-to-live value.

Remarks

When the message is first created, the body of the message is in write-only mode. After a message has been sent, the client that sent it can retain and modify it without affecting the message that has been sent. The same message object can be sent multiple times.

When a message is received, the provider has called QATextMessage.reset() so that the message body is in read-only mode and reading values starts from the beginning of the message body.

See also

- "QAMessageListener.onMessage method [QAnywhere Java]" on page 567

getText method

Returns the message text.

Syntax

```
String QATextMessage.getText() throws QAException
```

Returns

The message text, or null .

Exceptions

- **QAException class** Thrown if there is a problem retrieving the message text.

Remarks

If the message text exceeds the maximum size specified by the QAManager.MAX_IN_MEMORY_MESSAGE_SIZE property, this method returns null. In this case, use the QATextMessage.readText(int) method to read the text.

See also

- "QATextMessage.readText method [QAnywhere Java]" on page 572

getTextLength method

Returns the length, in characters, of the message.

Syntax

```
long QATextMessage.getTextLength() throws QAException
```

Returns

The length in characters of the message.

Exceptions

- **QAException class** Thrown if there is a problem retrieving the length of the message.

readText method

Returns unread text from the message.

Syntax

```
String QATextMessage.readText(int maxLength) throws QAException
```

Parameters

- **maxLength** The maximum number of characters to read.

Returns

The text.

Exceptions

- **QAException class** Thrown if there is a problem retrieving the unread text.

Remarks

Any additional unread text must be read by subsequent calls to this method. Text is read from the beginning of any unread text.

reset method

Resets the text position of the message to the beginning.

Syntax

```
void QATextMessage.reset() throws QAException
```

Exceptions

- **QAException class** Thrown if there is a problem resetting the text position of the message.

setText method

Overwrites the message text.

Syntax

```
void QATextMessage.setText(String value) throws QAException
```

Parameters

- **value** The text to write to the message body.

Exceptions

- **QAException class** Thrown if there is a problem overwriting the message text.

writeText method

Appends text to the text of the message.

Overload list

Name	Description
writeText(String) method	Appends text to the text of the message.
writeText(String, int) method	Appends text to the text of the message.
writeText(String, int, int) method	Appends text to the text of the message.

writeText(String) method

Appends text to the text of the message.

Syntax

```
void QATextMessage.writeText(String value) throws QAException
```

Parameters

- **value** The text to append.

Exceptions

- **QAException class** Thrown if there is a problem appending the message text.

writeText(String, int) method

Appends text to the text of the message.

Syntax

```
void QATextMessage.writeText(
     String value,
    int length
) throws QAException
```

Parameters

- **value** The text to append.
- **length** The number of characters of text to append.

Exceptions

- **QAException class** Thrown if there is a problem appending the message text.

writeText(String, int, int) method

Appends text to the text of the message.

Syntax

```
void QATextMessage.writeText(
     String value,
    int offset,
    int length
) throws QAException
```

Parameters

- **value** The text to append.
- **offset** The offset into value of the text to append.
- **length** The number of characters of text to append.

Exceptions

- **QAException class** Thrown if there is a problem appending the message text.

QATransactionalManager interface

The QATransactionalManager class derives from QAManagerBase and manages transactional QAnywhere messaging operations.

Syntax

```
public interface QATransactionalManager
```

Base classes

- "QAManagerBase interface [QAnywhere Java]" on page 514

Members

All members of QATransactionalManager interface, including all inherited members.

Name	Description
browseMessages method	Browses all available messages in the message store.
browseMessagesByID method	Browse the message with the given message ID.
browseMessagesByQueue method	Browses the available messages waiting that have been sent to the given address.
browseMessagesBySelector method	Browse messages queued in the message store that satisfy the given selector.
cancelMessage method	Cancels the message with the given message ID.

Name	Description
close method	Closes the connection to the QAnywhere message system and releases any resources used by the QAManagerBase.
commit method	Commits the current transaction and begins a new transaction.
createBinaryMessage method	Creates a QABinaryMessage object.
createTextMessage method	Creates a QATextMessage object.
getBooleanStoreProperty method	Gets a boolean value for a predefined or custom message store property.
getByteStoreProperty method	Gets a signed byte value for a predefined or custom message store property.
getDoubleStoreProperty method	Gets a double value for a predefined or custom message store property.
getFloatStoreProperty method	Gets a float value for a predefined or custom message store property.
getIntStoreProperty method	Gets an int value for a predefined or custom message store property.
getLongStoreProperty method	Gets a long value for a predefined or custom message store property.
getMessage method	Returns the next available QAMessage sent to the specified address.
getMessageBySelector method	Returns the next available QAMessage sent to the specified address that satisfies the given selector.
getMessageBySelectorNoWait method	Returns the next available QAMessage sent to the given address that satisfies the given selector.
getMessageBySelectorTimeout method	Returns the next available QAMessage sent to the given address that satisfies the given selector.
getMessageNoWait method	Returns the next available QAMessage sent to the given address.
getMessageTimeout method	Returns the next available QAMessage sent to the given address.
getMode method	Returns the QAManager acknowledgement mode for received messages.
getQueueDepth method	Returns the total depth of all queues, based on a given filter.

Name	Description
getShortStoreProperty method	Gets a short value for a predefined or custom message store property.
getStoreProperty method	Gets an Object representing a message store property.
getStorePropertyNames method	Gets an enumerator over the message store property names.
getStringStoreProperty method	Gets a string value for a predefined or custom message store property.
open method	Opens a QATransactionalManager instance.
propertyExists method	Tests if there currently exists a value for the given the property.
putMessage method	Prepares a message to send to another QAnywhere client.
putMessageTimeToLive method	Prepares a message to send to another QAnywhere client.
reOpen method	Reopens the QAManagerBase.
rollback method	Rolls back the current transaction and begins a new transaction.
setBooleanStoreProperty method	Sets a predefined or custom message store property to a boolean value.
setByteStoreProperty method	Sets a predefined or custom message store property to a sbyte value.
setDoubleStoreProperty method	Sets a predefined or custom message store property to a double value.
setFloatStoreProperty method	Sets a predefined or custom message store property to a float value.
setIntStoreProperty method	Sets a predefined or custom message store property to an int value.
setLongStoreProperty method	Sets a predefined or custom message store property to a long value.
setMessageListener method	Registers a QAMessageListener object to receive QAnywhere messages asynchronously.
setMessageListener2 method	Registers a QAMessageListener2 object to receive QAnywhere messages asynchronously.

Name	Description
setMessageListenerBySelector method	Registers a QAMessageListener object to receive QAnywhere messages asynchronously, with a message selector.
setMessageListenerBySelector2 method	Registers a QAMessageListener2 object to receive QAnywhere messages asynchronously, with a message selector.
setProperty method	Allows you to set QAnywhere Manager configuration properties programmatically.
setShortStoreProperty method	Sets a predefined or custom message store property to a short value.
setStoreProperty method	Sets a predefined or custom message store property to a System.Object value.
setStringStoreProperty method	Sets a predefined or custom message store property to a String value.
start method	Starts the QAManagerBase for receiving incoming messages.
stop method	Halts the QAManagerBase's reception of incoming messages.
triggerSendReceive method	Causes a synchronization with the QAnywhere message server, uploading any messages addressed to other clients, and downloading any messages addressed to the local client.

Remarks

For a detailed description of derived behavior, see QAManagerBase.

QATransactionalManager instances can only be used for transactional acknowledgement. Use the QATransactionalManager.commit() method to commit all QAManagerBase.putMessage(String, QAMessage) and QAManagerBase.getMessage(String) invocations.

See also

- “QATransactionalManager.commit method [QAnywhere Java]” on page 578
- “QAManagerBase.putMessage method [QAnywhere Java]” on page 534
- “QAManagerBase.getMessage method [QAnywhere Java]” on page 525

commit method

Commits the current transaction and begins a new transaction.

Syntax

```
void QATransactionalManager.commit() throws QAException
```


Exceptions

- **QAException class** Thrown if there is a problem committing.

Remarks

This method commits all QAManagerBase.putMessage(String, QAMessage) and QAManagerBase.getMessage(String) invocations.

> **Note**
> The first transaction begins with the call to QATransactionalManager.open().

open method

Opens a QATransactionalManager instance.

Syntax

```
void QATransactionalManager.open() throws QAException
```

Exceptions

- **QAException class** Thrown if there is a problem opening the manager.

Remarks

This method must be the first method called after creating a manager.

rollback method

Rolls back the current transaction and begins a new transaction.

Syntax

```
void QATransactionalManager.rollback() throws QAException
```

Exceptions

- **QAException class** Thrown if there is a problem rolling back.

Remarks

This method rolls back all uncommitted QAManagerBase.putMessage(String, QAMessage) and QAManagerBase.getMessage(String) invocations.

QueueDepthFilter interface

Provides queue depth filter values for QAManagerBase.getQueueDepth(short) and QAManagerBase.getQueueDepth(String, short).

Syntax

```
public interface QueueDepthFilter
```

Members

All members of QueueDepthFilter interface, including all inherited members.

Name	Description
ALL variable	This filter specifies both incoming and outgoing messages.
INCOMING variable	This filter specifies only incoming messages.
LOCAL variable	This filter specifies only local messages.
OUTGOING variable	This filter specifies only outgoing messages.

See also

- "QAManagerBase.getQueueDepth method [QAnywhere Java]" on page 530

ALL variable

This filter specifies both incoming and outgoing messages.

Syntax

```
final short QueueDepthFilter.ALL
```

Remarks

System messages and expired messages are not included in any queue depth counts.

INCOMING variable

This filter specifies only incoming messages.

Syntax

```
final short QueueDepthFilter.INCOMING
```

Remarks

An incoming message is defined as a message whose originator is different than the agent ID of the message store.

LOCAL variable

This filter specifies only local messages.

Syntax

```
final short QueueDepthFilter.LOCAL
```

Remarks

A local message is defined as a message whose originator and target are the agent ID of the message store.

OUTGOING variable

This filter specifies only outgoing messages.

Syntax

```
final short QueueDepthFilter.OUTGOING
```

Remarks

An outgoing message is defined as a message whose originator is the agent ID of the message store, and whose destination is not the agent ID of the message store.

StatusCodes interface

This interface defines a set of codes for the status of a message.

Syntax

```
public interface StatusCodes
```

Members

All members of StatusCodes interface, including all inherited members.

Name	Description
CANCELED variable	The message has been canceled.
EXPIRED variable	The message has expired; the message was not received before its expiration time had passed.
FINAL variable	This constant is used to determine if a message has achieved a final state.
LOCAL variable	The message is addressed to the local message store and will not be transmitted to the server.

Name	Description
PENDING variable	The message has been sent but not received and acknowledged.
RECEIVED variable	The message has been received and acknowledged by the receiver.
RECEIVING variable	The message is in the process of being received, or it was received but not acknowledged.
TRANSMITTED variable	The message has been transmitted to the server.
TRANSMITTING variable	The message is in the process of being transmitted to the server.
UNRECEIVABLE variable	The message has been marked as unreceivable.
UNTRANSMITTED variable	The message has not been transmitted to the server.

CANCELED variable

The message has been canceled.

Syntax

```
final int StatusCodes.CANCELED
```

Remarks

This code applies to MessageProperties.STATUS.

See also

- "MessageProperties.STATUS variable [QAnywhere Java]" on page 475

EXPIRED variable

The message has expired; the message was not received before its expiration time had passed.

Syntax

```
final int StatusCodes.EXPIRED
```

Remarks

This code applies to MessageProperties.STATUS.

See also

- "MessageProperties.STATUS variable [QAnywhere Java]" on page 475

FINAL variable

This constant is used to determine if a message has achieved a final state.

Syntax

```
final int StatusCodes.FINAL
```

Remarks

A message has achieved a final state if and only if its status is greater than this constant.

This code applies to MessageProperties.STATUS.

See also

- "MessageProperties.STATUS variable [QAnywhere Java]" on page 475

LOCAL variable

The message is addressed to the local message store and will not be transmitted to the server.

Syntax

```
final int StatusCodes.LOCAL
```

Remarks

This code applies to MessageProperties.TRANSMISSION_STATUS.

See also

- "MessageProperties.TRANSMISSION_STATUS variable [QAnywhere Java]" on page 476

PENDING variable

The message has been sent but not received and acknowledged.

Syntax

```
final int StatusCodes.PENDING
```

Remarks

This code applies to MessageProperties.STATUS.

See also

- "MessageProperties.STATUS variable [QAnywhere Java]" on page 475

RECEIVED variable

The message has been received and acknowledged by the receiver.

Syntax

```
final int StatusCodes.RECEIVED
```

Remarks

This code applies to MessageProperties.STATUS.

See also

- "MessageProperties.STATUS variable [QAnywhere Java]" on page 475

RECEIVING variable

The message is in the process of being received, or it was received but not acknowledged.

Syntax

```
final int StatusCodes.RECEIVING
```

Remarks

This code applies to MessageProperties.STATUS.

See also

- "MessageProperties.STATUS variable [QAnywhere Java]" on page 475

TRANSMITTED variable

The message has been transmitted to the server.

Syntax

```
final int StatusCodes.TRANSMITTED
```

Remarks

This code applies to MessageProperties.TRANSMISSION_STATUS.

See also

- "MessageProperties.TRANSMISSION_STATUS variable [QAnywhere Java]" on page 476

TRANSMITTING variable

The message is in the process of being transmitted to the server.

Syntax

```
final int StatusCodes.TRANSMITTING
```

Remarks

This code applies to MessageProperties.TRANSMISSION_STATUS.

See also

- "MessageProperties.TRANSMISSION_STATUS variable [QAnywhere Java]" on page 476

UNRECEIVABLE variable

The message has been marked as unreceivable.

Syntax

```
final int StatusCodes.UNRECEIVABLE
```

Remarks

The message is either malformed, or there were too many failed attempts to deliver it.

This code applies to MessageProperties.STATUS.

See also

- "MessageProperties.STATUS variable [QAnywhere Java]" on page 475

UNTRANSMITTED variable

The message has not been transmitted to the server.

Syntax

```
final int StatusCodes.UNTRANSMITTED
```

Remarks

This code applies to MessageProperties.TRANSMISSION_STATUS.

See also

- "MessageProperties.TRANSMISSION_STATUS variable [QAnywhere Java]" on page 476

QAnywhere Java API reference for web services

Package

```
ianywhere.qanywhere.ws
```

WSBase class

This is the base class for the main web service proxy class generated by the mobile web service compiler.

Syntax

```
public class WSBase
```

Members

All members of WSBase class, including all inherited members.

Name	Description
WSBase constructor	Constructor with configuration property file.
clearRequestProperties method	Clears all request properties that have been set for this WSBase.
getResult method	Gets a WSResult object that represents the results of a web service request.
getServiceID method	Gets the service ID for this instance of WSBase.
setListener method	Sets a listener for the results of a given web service request.
setProperty method	Sets a configuration property for this instance of WSBase.
setQAManager method	Sets the QAManagerBase that is used by this web service client to do web service requests.
setRequestProperty method	Sets a request property for webservice requests made by this instance of WSBase.
setServiceID method	Sets a user-defined ID for this instance of WSBase.

WSBase constructor

Constructor with configuration property file.

Overload list

Name	Description
WSBase() constructor	Constructor.
WSBase(String) constructor	Constructor with configuration property file.

WSBase() constructor

Constructor.

Syntax

```
WSBase.WSBase() throws WSException
```

Exceptions

- **WSException class** Thrown if there is a problem constructing the WSBase.

WSBase(String) constructor

Constructor with configuration property file.

Syntax

```
WSBase.WSBase(String iniFile) throws WSException
```

Parameters

- **iniFile** A file containing configuration properties.

Exceptions

- **WSException class** Thrown if there is a problem constructing the WSBase.

Remarks

Valid configuration properties are:

LOG_FILE a file to which to log runtime information.

LOG_LEVEL a value between 0 and 6 that controls the verbosity of information logged, with 6 being the highest verbosity.

WS_CONNECTOR_ADDRESS the address of the web service connector in the MobiLink server. The default WS_CONNECTOR_ADDRESS is "ianywhere.connector.webservices\\".

clearRequestProperties method

Clears all request properties that have been set for this WSBase.

Syntax

```
void WSBase.clearRequestProperties()
```

getResult method

Gets a WSResult object that represents the results of a web service request.

Syntax

```
WSResult WSBase.getResult(String requestID) throws QAException
```

Parameters

- **requestID** The ID of the web service request.

Returns

A WSResult instance representing the results of the web service request.

See also

- "WSStatus class [QAnywhere Java]" on page 617

getServiceID method

Gets the service ID for this instance of WSBase.

Syntax

```
String WSBase.getServiceID()
```

Returns

The service ID.

setListener method

Sets a listener for the results of a given web service request.

Overload list

Name	Description
setListener(String, WSListener) method	Sets a listener for the results of a given web service request.
setListener(WSListener) method	Sets a listener for the results of all web service requests made by this instance of WSBase.

setListener(String, WSListener) method

Sets a listener for the results of a given web service request.

Syntax

```
void WSBase.setListener(
     String requestID,
    WSListener listener
) throws QAException
```

Parameters

- **requestID** The ID of the web service request to which to listen for results.
- **listener** The listener object that gets called when the result of the given web service request is available.

Remarks

Listeners are typically used to get results of the asyncXYZ methods of the service.

To remove a listener, call setListener with *null* as the listener.

Note
This method replaces the listener set by any previous call to setListener.

setListener(WSListener) method

Sets a listener for the results of all web service requests made by this instance of WSBase.

Syntax

```
void WSBase.setListener(WSListener listener) throws QAException
```

Parameters

- **listener** The listener object that gets called when the result of a web service request is available.

Remarks

Listeners are typically used to get results of the asyncXYZ methods of the service.

To remove a listener, call setListener with *null* as the listener.

Note
This method replaces the listener set by any previous call to setListener.

setProperty method

Sets a configuration property for this instance of WSBase.

Syntax

```
void WSBase.setProperty(String property, String val) throws WSException
```

Parameters

- **property** The property name to set.
- **val** The property value.

Remarks

Configuration properties must be set before any asynchronous or synchronous web service request is made; after which this method has no effect.

Valid configuration properties are:

LOG_FILE a file to which to log runtime information.

LOG_LEVEL a value between 0 and 6 that controls the verbosity of information logged, with 6 being the highest verbosity.

WS_CONNECTOR_ADDRESS the address of the web service connector in the MobiLink server. The default is: "ianywhere.connector.webservices\\".

setQAManager method

Sets the QAManagerBase that is used by this web service client to do web service requests.

Syntax

```
void WSBase.setQAManager(QAManagerBase mgr)
```

Parameters

- **mgr** The QAManagerBase to use.

Remarks

Note
If you use an EXPLICIT_ACKNOWLEDGEMENT QAManager, you can acknowledge the result of an asynchronous web service request by calling the acknowledge() method of WSResult. The result of a synchronous web service request is automatically acknowledged, even in the case of an EXPLICIT_ACKNOWLEDGEMENT QAManager. If you use an IMPLICIT_ACKNOWLEDGEMENT QAManager, the result of any web service request is acknowledged automatically.

setRequestProperty method

Sets a request property for webservice requests made by this instance of WSBase.

Syntax

```
void WSBase.setRequestProperty(String name, Object value)
```

Parameters

- **name** The property name to set.
- **value** The property value.

Remarks

A request property is set on each QAMessage that is sent by this WSBase, until the property is cleared. A request property is cleared by setting it to a null value. The type of the message property is determined by the class of the value parameter. For example, if value is an instance of Integer, then setIntProperty is used to set the property on the QAMessage.

setServiceID method

Sets a user-defined ID for this instance of WSBase.

Syntax

```
void WSBase.setServiceID(String serviceID)
```

Parameters

- **serviceID** The service ID.

Remarks

The service ID should be set to a value unique to this instance of WSBase. It is used internally to form a queue name for sending and receiving web service requests. Therefore, the service ID should be persisted between application sessions, in order to retrieve results of web service requests made in a previous session.

WSException class

This class represents an exception that occurred during processing of a web service request.

Syntax

```
public class WSException
```

Derived classes

- "WSFaultException class [QAnywhere Java]" on page 594

Members

All members of WSException class, including all inherited members.

Name	Description
WSException constructor	Constructs a new exception with the specified error message.
getErrorCode method	Gets the error code associated with this exception.
WS_STATUS_HTTP_ERROR variable	Error code indicating that there was an error in the web service HTTP request made by the web services connector.
WS_STATUS_HTTP_OK variable	Error code indicating that the webservice HTTP request by the web services connector was successful.
WS_STATUS_HTTP_RETRIES_EXCEEDED variable	Error code indicating that the number of HTTP retries was exceeded the web services connector.
WS_STATUS_SOAP_PARSE_ERROR variable	Error code indicating that there was an error in the web services runtime or in the webservices connector in parsing a SOAP response or request.

WSException constructor

Constructs a new exception with the specified error message.

Overload list

Name	Description
WSException(Exception) constructor	Constructs a new exception.
WSException(String) constructor	Constructs a new exception with the specified error message.
WSException(String, int) constructor	Constructs a new exception with the specified error message and error code.

WSException(Exception) constructor

Constructs a new exception.

Syntax

```
WSException.WSException(Exception exception)
```

Parameters

- **exception** The exception.

WSException(String) constructor

Constructs a new exception with the specified error message.

Syntax

```
WSException.WSException(String msg)
```

Parameters

- **msg** The error message.

WSException(String, int) constructor

Constructs a new exception with the specified error message and error code.

Syntax

```
WSException.WSException(String msg, int errorCode)
```

Parameters

- **msg** The error message.
- **errorCode** The error code.

getErrorCode method

Gets the error code associated with this exception.

Syntax

```
int WSException.getErrorCode()
```

Returns

The error code associated with this exception.

WS_STATUS_HTTP_ERROR variable

Error code indicating that there was an error in the web service HTTP request made by the web services connector.

Syntax

```
final int WSException.WS_STATUS_HTTP_ERROR
```

WS_STATUS_HTTP_OK variable

Error code indicating that the webservice HTTP request by the web services connector was successful.

Syntax

```
final int WSException.WS_STATUS_HTTP_OK
```

WS_STATUS_HTTP_RETRIES_EXCEEDED variable

Error code indicating that the number of HTTP retries was exceeded the web services connector.

Syntax

```
final int WSException.WS_STATUS_HTTP_RETRIES_EXCEEDED
```

WS_STATUS_SOAP_PARSE_ERROR variable

Error code indicating that there was an error in the web services runtime or in the webservices connector in parsing a SOAP response or request.

Syntax

```
final int WSException.WS_STATUS_SOAP_PARSE_ERROR
```

WSFaultException class

This class represents a SOAP Fault exception from the web service connector.

Syntax

```
public class WSFaultException
```

Base classes

- "WSException class [QAnywhere Java]" on page 591

Members

All members of WSFaultException class, including all inherited members.

Name	Description
WSException constructor	Constructs a new exception with the specified error message.
WSFaultException constructor	Constructs a new exception with the specified error message.
getErrorCode method	Gets the error code associated with this exception.
WS_STATUS_HTTP_ERROR variable	Error code indicating that there was an error in the web service HTTP request made by the web services connector.

Name	Description
WS_STATUS_HTTP_OK variable	Error code indicating that the webservice HTTP request by the web services connector was successful.
WS_STATUS_HTTP_RETRIES_EXCEEDED variable	Error code indicating that the number of HTTP retries was exceeded the web services connector.
WS_STATUS_SOAP_PARSE_ERROR variable	Error code indicating that there was an error in the web services runtime or in the webservices connector in parsing a SOAP response or request.

WSFaultException constructor

Constructs a new exception with the specified error message.

Syntax

```
WSFaultException.WSFaultException(String msg)
```

Parameters

- **msg** The error message.

WSListener interface

This class represents a listener for results of web service requests.

Syntax

```
public interface WSListener
```

Members

All members of WSListener interface, including all inherited members.

Name	Description
onException method	Called when an exception occurs during processing of the result of an asynchronous web service request.
onResult method	Called with the result of an asynchronous web service request.

onException method

Called when an exception occurs during processing of the result of an asynchronous web service request.

Syntax

```
void WSListener.onException(WSException e, WSResult wsResult)
```

Parameters

- **e** The WSException that occurred during processing of the result.
- **wsResult** A WSResult, from which the request ID may be obtained. Values of this WSResult are not defined.

See also

- "WSException class [QAnywhere Java]" on page 591
- "WSResult class [QAnywhere Java]" on page 596

onResult method

Called with the result of an asynchronous web service request.

Syntax

```
void WSListener.onResult(WSResult wsResult)
```

Parameters

- **wsResult** The WSResult describing the result of a web service request.

See also

- "WSResult class [QAnywhere Java]" on page 596

WSResult class

This class represents the results of a web service request.

Syntax

```
public class WSResult
```

Members

All members of WSResult class, including all inherited members.

Name	Description
acknowledge method	Acknowledges that this WSResult has been processed.
getArrayValue method	Gets an array of complex types value from this WSResult.
getBigDecimalArrayValue method	Gets a BigDecimal array value from this WSResult.

Name	Description
getBigDecimalValue method	Gets a BigDecimal value from this WSResult.
getBigIntegerArrayValue method	Gets a BigInteger array value from this WSResult.
getBigIntegerValue method	Gets a BigInteger value from this WSResult.
getBooleanArrayValue method	Gets a java.lang.Boolean array value from this WSResult.
getBooleanValue method	Gets a java.lang.Boolean value from this WSResult.
getByteArrayValue method	Gets a java.lang.Byte array value from this WSResult.
getByteValue method	Gets a java.lang.Byte value from this WSResult.
getCharacterArrayValue method	Gets a java.lang.Character array value from this WSResult.
getCharacterValue method	Gets a java.lang.Character value from this WSResult.
getDoubleArrayValue method	Gets a java.lang.Double array value from this WSResult.
getDoubleValue method	Gets a java.lang.Double value from this WSResult.
getErrorMessage method	Gets the error message.
getFloatArrayValue method	Gets a java.lang.Float array value from this WSResult.
getFloatValue method	Gets a java.lang.Float value from this WSResult.
getIntegerArrayValue method	Gets a java.lang.Integer array value from this WSResult.
getIntegerValue method	Gets a java.lang.Integer value from this WSResult.
getLongArrayValue method	Gets a java.lang.Long array value from this WSResult.
getLongValue method	Gets a java.lang.Long value from this WSResult.
getObjectArrayValue method	Gets an array of complex types value from this WSResult.
getObjectValue method	Gets value of a complex type from this WSResult.
getPrimitiveBooleanArrayValue method	Gets a boolean array value from this WSResult.
getPrimitiveBooleanValue method	Gets a boolean value from this WSResult.
getPrimitiveByteArrayValue method	Gets a byte array value from this WSResult.
getPrimitiveByteValue method	Gets a byte value from this WSResult.

Name	Description
getPrimitiveCharArrayValue method	Gets a char array value from this WSResult.
getPrimitiveCharValue method	Gets a char value from this WSResult.
getPrimitiveDoubleArrayValue method	Gets a double array value from this WSResult.
getPrimitiveDoubleValue method	Gets a double value from this WSResult.
getPrimitiveFloatArrayValue method	Gets a float array value from this WSResult.
getPrimitiveFloatValue method	Gets a float value from this WSResult.
getPrimitiveIntArrayValue method	Gets an int array value from this WSResult.
getPrimitiveIntValue method	Gets an int value from this WSResult.
getPrimitiveLongArrayValue method	Gets a long array value from this WSResult.
getPrimitiveLongValue method	Gets a long value from this WSResult.
getPrimitiveShortArrayValue method	Gets a short array value from this WSResult.
getPrimitiveShortValue method	Gets a short value from this WSResult.
getRequestID method	Gets the request ID that this WSResult represents.
getShortArrayValue method	Gets a java.lang.Short array value from this WSResult.
getShortValue method	Gets a java.lang.Short value from this WSResult.
getStatus method	Gets the status of this WSResult.
getStringArrayValue method	Gets a String array value from this WSResult.
getStringValue method	Gets a String value from this WSResult.
getValue method	Gets the value of a complex type from this WSResult.

Remarks

A WSResult object is obtained in one of three ways:

- It is passed to the WSListener.onResult.
- It is returned by an asyncXYZ method of the service proxy generated by the compiler.
- It is obtained by calling WSBase.getResult with a specific request ID.

acknowledge method

Acknowledges that this WSResult has been processed.

Syntax

```
void WSResult.acknowledge() throws WSException, QAException
```

Remarks

This method is only useful when an EXPLICIT_ACKNOWLEDGEMENT QAManager is being used.

getArrayValue method

Gets an array of complex types value from this WSResult.

Syntax

```
WSSerializable[] WSResult.getArrayValue(
     String parentName
) throws WSException
```

Parameters

- **parentName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **WSException class** Thrown if there is a problem getting the value.

getBigDecimalArrayValue method

Gets a BigDecimal array value from this WSResult.

Syntax

```
BigDecimal[] WSResult.getBigDecimalArrayValue(
     String elementName
) throws WSException
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **WSException class** Thrown if there is a problem getting the value.

getBigDecimalValue method

Gets a BigDecimal value from this WSResult.

Syntax

```
BigDecimal WSResult.getBigDecimalValue(
     String elementName
) throws WSException
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **WSException class** Thrown if there is a problem getting the value.

getBigIntegerArrayValue method

Gets a BigInteger array value from this WSResult.

Syntax

```
BigInteger[] WSResult.getBigIntegerArrayValue(
     String elementName
) throws WSException
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **WSException class** Thrown if there is a problem getting the value.

getBigIntegerValue method

Gets a BigInteger value from this WSResult.

Syntax

```
BigInteger WSResult.getBigIntegerValue(
     String elementName
) throws WSException
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **WSException class** Thrown if there is a problem getting the value.

getBooleanArrayValue method

Gets a java.lang.Boolean array value from this WSResult.

Syntax

```
Boolean[] WSResult.getBooleanArrayValue(
     String elementName
) throws WSException
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **WSException class** Thrown if there is a problem getting the value.

getBooleanValue method

Gets a java.lang.Boolean value from this WSResult.

Syntax

```
Boolean WSResult.getBooleanValue(String elementName) throws WSException
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **WSException class** Thrown if there is a problem getting the value.

getByteArrayValue method

Gets a java.lang.Byte array value from this WSResult.

Syntax

```
Byte[] WSResult.getByteArrayValue(String elementName) throws WSException
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **WSException class** Thrown if there is a problem getting the value.

getByteValue method

Gets a java.lang.Byte value from this WSResult.

Syntax

```
Byte WSResult.getByteValue(String elementName) throws WSException
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **WSException class** Thrown if there is a problem getting the value.

getCharacterArrayValue method

Gets a java.lang.Character array value from this WSResult.

Syntax

```
Character[] WSResult.getCharacterArrayValue(
     String elementName
) throws WSException
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **WSException class** Thrown if there is a problem getting the value.

getCharacterValue method

Gets a java.lang.Character value from this WSResult.

Syntax

```
Character WSResult.getCharacterValue(
     String elementName
) throws WSException
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **WSException class** Thrown if there is a problem getting the value.

getDoubleArrayValue method

Gets a java.lang.Double array value from this WSResult.

Syntax

```
Double[] WSResult.getDoubleArrayValue(
     String elementName
) throws WSException
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **WSException class** Thrown if there is a problem getting the value.

getDoubleValue method

Gets a java.lang.Double value from this WSResult.

Syntax

```
Double WSResult.getDoubleValue(String elementName) throws WSException
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **WSException class** Thrown if there is a problem getting the value.

getErrorMessage method

Gets the error message.

Syntax

```
String WSResult.getErrorMessage()
```

Returns

The error message.

getFloatArrayValue method

Gets a java.lang.Float array value from this WSResult.

Syntax

```
Float[] WSResult.getFloatArrayValue(
     String elementName
) throws WSException
```


Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **WSException class** Thrown if there is a problem getting the value.

getFloatValue method

Gets a java.lang.Float value from this WSResult.

Syntax

```
Float WSResult.getFloatValue(String elementName) throws WSException
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **WSException class** Thrown if there is a problem getting the value.

getIntegerArrayValue method

Gets a java.lang.Integer array value from this WSResult.

Syntax

```
Integer[] WSResult.getIntegerArrayValue(
     String elementName
) throws WSException
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **WSException class** Thrown if there is a problem getting the value.

getIntegerValue method

Gets a java.lang.Integer value from this WSResult.

Syntax

```
Integer WSResult.getIntegerValue(String elementName) throws WSException
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **WSException class** Thrown if there is a problem getting the value.

getLongArrayValue method

Gets a java.lang.Long array value from this WSResult.

Syntax

```
Long[] WSResult.getLongArrayValue(String elementName) throws WSException
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **WSException class** Thrown if there is a problem getting the value.

getLongValue method

Gets a java.lang.Long value from this WSResult.

Syntax

```
Long WSResult.getLongValue(String elementName) throws WSException
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **WSException class** Thrown if there is a problem getting the value.

getObjectArrayValue method

Gets an array of complex types value from this WSResult.

Syntax

```
Object[] WSResult.getObjectArrayValue(
     String elementName
) throws WSException
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **WSException class** Thrown if there is a problem getting the value.

getObjectValue method

Gets value of a complex type from this WSResult.

Syntax

```
Object WSResult.getObjectValue(String elementName) throws WSException
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **WSException class** Thrown if there is a problem getting the value.

getPrimitiveBooleanArrayValue method

Gets a boolean array value from this WSResult.

Syntax

```
boolean[] WSResult.getPrimitiveBooleanArrayValue(
     String elementName
) throws WSException
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **WSException class** Thrown if there is a problem getting the value.

getPrimitiveBooleanValue method

Gets a boolean value from this WSResult.

Syntax

```
boolean WSResult.getPrimitiveBooleanValue(
     String elementName
) throws WSException
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **WSException class** Thrown if there is a problem getting the value.

getPrimitiveByteArrayValue method

Gets a byte array value from this WSResult.

Syntax

```
byte[] WSResult.getPrimitiveByteArrayValue(
     String elementName
) throws WSException
```


Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **WSException class** Thrown if there is a problem getting the value.

getPrimitiveByteValue method

Gets a byte value from this WSResult.

Syntax

```
byte WSResult.getPrimitiveByteValue(
     String elementName
) throws WSException
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **WSException class** Thrown if there is a problem getting the value.

getPrimitiveCharArrayValue method

Gets a char array value from this WSResult.

Syntax

```
char[] WSResult.getPrimitiveCharArrayValue(
     String elementName
) throws WSException
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **WSException class** Thrown if there is a problem getting the value.

getPrimitiveCharValue method

Gets a char value from this WSResult.

Syntax

```
char WSResult.getPrimitiveCharValue(
     String elementName
) throws WSException
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **WSException class** Thrown if there is a problem getting the value.

getPrimitiveDoubleArrayValue method

Gets a double array value from this WSResult.

Syntax

```
double[] WSResult.getPrimitiveDoubleArrayValue(
     String elementName
) throws WSException
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **WSException class** Thrown if there is a problem getting the value.

getPrimitiveDoubleValue method

Gets a double value from this WSResult.

Syntax

```
double WSResult.getPrimitiveDoubleValue(
     String elementName
) throws WSException
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **WSException class** Thrown if there is a problem getting the value.

getPrimitiveFloatArrayValue method

Gets a float array value from this WSResult.

Syntax

```
float[] WSResult.getPrimitiveFloatArrayValue(
     String elementName
) throws WSException
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **WSException class** Thrown if there is a problem getting the value.

getPrimitiveFloatValue method

Gets a float value from this WSResult.

Syntax

```
float WSResult.getPrimitiveFloatValue(
     String elementName
) throws WSException
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **WSException class** Thrown if there is a problem getting the value.

getPrimitiveIntArrayValue method

Gets an int array value from this WSResult.

Syntax

```
int[] WSResult.getPrimitiveIntArrayValue(
     String elementName
) throws WSException
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **WSException class** Thrown if there is a problem getting the value.

getPrimitiveIntValue method

Gets an int value from this WSResult.

Syntax

```
int WSResult.getPrimitiveIntValue(String elementName) throws WSException
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **WSException class** Thrown if there is a problem getting the value.

getPrimitiveLongArrayValue method

Gets a long array value from this WSResult.

Syntax

```
long[] WSResult.getPrimitiveLongArrayValue(
     String elementName
) throws WSException
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **WSException class** Thrown if there is a problem getting the value.

getPrimitiveLongValue method

Gets a long value from this WSResult.

Syntax

```
long WSResult.getPrimitiveLongValue(
     String elementName
) throws WSException
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **WSException class** Thrown if there is a problem getting the value.

getPrimitiveShortArrayValue method

Gets a short array value from this WSResult.

Syntax

```
short[] WSResult.getPrimitiveShortArrayValue(
     String elementName
) throws WSException
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **WSException class** Thrown if there is a problem getting the value.

getPrimitiveShortValue method

Gets a short value from this WSResult.

Syntax

```
short WSResult.getPrimitiveShortValue(
     String elementName
) throws WSException
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **WSException class** Thrown if there is a problem getting the value.

getRequestID method

Gets the request ID that this WSResult represents.

Syntax

```
String WSResult.getRequestID()
```

Returns

The request ID.

Remarks

This request ID should be persisted between runs of the application if it is desired to obtain a WSResult corresponding to a web service request in a run of the application different from when the request was made.

getShortArrayValue method

Gets a java.lang.Short array value from this WSResult.

Syntax

```
Short[] WSResult.getShortArrayValue(
     String elementName
) throws WSException
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **WSException class** Thrown if there is a problem getting the value.

getShortValue method

Gets a java.lang.Short value from this WSResult.

Syntax

```
Short WSResult.getShortValue(String elementName) throws WSException
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **WSException class** Thrown if there is a problem getting the value.

getStatus method

Gets the status of this WSResult.

Syntax

```
int WSResult.getStatus() throws QAException, WSException
```

Returns

The status code.

See also

- "WSStatus class [QAnywhere Java]" on page 617

getStringArrayValue method

Gets a String array value from this WSResult.

Syntax

```
String[] WSResult.getStringArrayValue(
     String elementName
) throws WSException
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **WSException class** Thrown if there is a problem getting the value.

getStringValue method

Gets a String value from this WSResult.

Syntax

```
String WSResult.getStringValue(String elementName) throws WSException
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **WSException class** Thrown if there is a problem getting the value.

getValue method

Gets the value of a complex type from this WSResult.

Syntax

```
Object WSResult.getValue(String elementName) throws WSException
```

Parameters

- **elementName** The element name in the WSDL document of this value.

Returns

The value.

Exceptions

- **WSException class** Thrown if there is a problem getting the value.

WSStatus class

This class defines codes for the status of a web service request.

Syntax

```
public class WSStatus
```

Members

All members of WSStatus class, including all inherited members.

Name	Description
STATUS_ERROR variable	There was an error processing the request.
STATUS_QUEUED variable	The request has been queued for delivery to the server.
STATUS_RESULT_AVAILABLE variable	The result of the request is available.
STATUS_SUCCESS variable	The request was successful.

STATUS_ERROR variable

There was an error processing the request.

Syntax

```
final int WSStatus.STATUS_ERROR
```

STATUS_QUEUED variable

The request has been queued for delivery to the server.

Syntax

```
final int WSStatus.STATUS_QUEUED
```

Remarks

The final status of the request is not known yet.

STATUS_RESULT_AVAILABLE variable

The result of the request is available.

Syntax

```
final int WSStatus.STATUS_RESULT_AVAILABLE
```

STATUS_SUCCESS variable

The request was successful.

Syntax

```
final int WSStatus.STATUS_SUCCESS
```

QAnywhere SQL API reference

Message properties, headers, and content

This section documents QAnywhere SQL stored procedures that help you set message headers, message content, and message properties.

Message headers

You can use the following stored procedures to get and set message header information.

See "Message headers" on page 656.

ml_qa_getaddress

Returns the QAnywhere address of the message.

Parameters

Item	Description	Remarks
1	Message ID	VARCHAR(128). You can obtain the message ID from ml_qa_createmessage or ml_qa_getmessage.

Return value

The QAnywhere message address as VARCHAR(128). QAnywhere message addresses take the form *id* *queue-name*.

You can read this header after a message is received and until a rollback or commit occurs; after that you cannot read it.

See also

- “SQL application setup” on page 56
- “QAnywhere message addresses” on page 57
- “ml_qa_createmessage” on page 648
- “ml_qa_getmessage” on page 649

Example

In the following example, a message is received and its address is output to the database server messages window:

```
begin
    declare @msgid varchar(128);
    declare @addr varchar(128);
    set @msgid = ml_qa_getmessage( 'myaddress' );
    set @addr = ml_qa_getaddress( @msgid );
    message 'message to address ' || @addr || ' received';
    commit;
end
```

ml_qa_getexpiration

Returns the expiration time of the message.

Parameters

Item	Description	Remarks
1	Message ID	VARCHAR(128). You can obtain the message ID from ml_qa_createmessage or ml_qa_getmessage.

Return value

The expiration time as TIMESTAMP. Returns null if there is no expiration.

Remarks

After completion of ml_qa_putmessage, a message expires if it is not received by the intended recipient in the specified time. The message may then be deleted using default QAnywhere delete rules.

You can read this header after a message is received and until a rollback or commit occurs; after that you cannot read it.

See also

- "SQL application setup" on page 56
- "Message delete rules" on page 741
- "Sending QAnywhere messages" on page 61
- "ml_qa_setexpiration" on page 625
- "ml_qa_createmessage" on page 648
- "ml_qa_getmessage" on page 649

Example

In the following example, a message is received and the message expiration is output to the database server messages window:

```
begin
    declare @msgid varchar(128);
    declare @expires timestamp;
    set @msgid = ml_qa_getmessage( 'myaddress' );
    set @expires = ml_qa_getexpiration( @msgid );
    message 'message would have expired at ' || @expires || ' if it had not
been received';
    commit;
end
```

ml_qa_getinreplytoid

Returns the in-reply-to ID for the message.

Parameters

Item	Description	Remarks
1	Message ID	VARCHAR(128). You can obtain the message ID from ml_qa_createmessage or ml_qa_getmessage.

Return value

The in-reply-to ID as VARCHAR(128).

Remarks

A client can use the InReplyToID header field to link one message with another. A typical use is to link a response message with its request message.

The in-reply-to ID is the ID of the message that this message is replying to.

You can read this header after a message is received and until a rollback or commit occurs; after that you cannot read it.

See also

- "SQL application setup" on page 56
- "ml_qa_setinreplytoid" on page 626
- "ml_qa_createmessage" on page 648
- "ml_qa_getmessage" on page 649

Example

In the following example, a message is received and the in-reply-to-id of the message is output to the database server messages window:

```
begin
    declare @msgid varchar(128);
    declare @inreplytoid varchar(128);
    set @msgid = ml_qa_getmessage( 'myaddress' );
    set @inreplytoid = ml_qa_getinreplytoid( @msgid );
    message 'message is likely a reply to the message with id ' ||
@inreplytoid;
    commit;
end
```

ml_qa_getpriority

Returns the priority level of the message.

Parameters

Item	Description	Remarks
1	Message ID	VARCHAR(128). You can obtain the message ID from ml_qa_createmessage or ml_qa_getmessage.

Return value

The priority level as INTEGER.

Remarks

The QAnywhere API defines ten levels of priority value, with 0 as the lowest priority and 9 as the highest. Clients should consider priorities 0-4 as gradations of normal priority and priorities 5-9 as gradations of expedited priority.

You can read this header after a message is received and until a rollback or commit occurs; after that you cannot read it.

See also

- "SQL application setup" on page 56
- "ml_qa_setpriority" on page 627
- "ml_qa_createmessage" on page 648
- "ml_qa_getmessage" on page 649

Example

In the following example, a message is received and the priority of the message is output to the database server messages window:

```
begin
    declare @msgid varchar(128);
    declare @priority integer;
    set @msgid = ml_qa_getmessage( 'myaddress' );
    set @priority = ml_qa_getpriority( @msgid );
    message 'a message with priority ' || @priority || ' has been received';
    commit;
end
```

ml_qa_getredelivered

Returns a value indicating whether this message has previously been received but not acknowledged.

Parameters

Item	Description	Remarks
1	Message ID	VARCHAR(128). You can obtain the message ID from ml_qa_createmessage or ml_qa_getmessage.

Return value

The redelivered value as BIT. A value of **1** indicates that the message is being redelivered; **0** indicates that it is not being redelivered.

Remarks

A message may be redelivered if it was previously received but not acknowledged. For example, the message was received but the application receiving the message did not complete processing the message content before it crashed. In these cases, QAnywhere marks the message as redelivered to alert the receiver that the message might be partly processed.

For example, assume that the receipt of a message occurs in three steps:

1. An application using a non-transactional QAnywhere manager receives the message.
2. The application writes the message content and message ID to a database table called T1, and commits the change.
3. The application acknowledges the message.

If the application fails between steps 1 and 2 or between steps 2 and 3, the message is redelivered when the application restarts.

If the failure occurs between steps 1 and 2, you should process the redelivered message by running steps 2 and 3. If the failure occurs between steps 2 and 3, then the message is already processed and you only need to acknowledge it.

To determine what happened when the application fails, you can have the application call ml_qa_getredelivered to check if the message has been previously redelivered. Only messages that are redelivered need to be looked up in table T1. This is more efficient than having the application access the received message's message ID to check whether the message is in the table T1, because application failures are rare.

You can read this header after a message is received and until a rollback or commit occurs; after that you cannot read it.

See also

- "SQL application setup" on page 56
- "ml_qa_createmessage" on page 648
- "ml_qa_getmessage" on page 649

Example

In the following example, a message is received; if the message was previously delivered but not received, the message ID is output to the database server messages window:

```
begin
    declare @msgid varchar(128);
    declare @redelivered bit;
    set @msgid = ml_qa_getmessage( 'myaddress' );
    set @redelivered = ml_qa_getredelivered( @msgid );
    if @redelivered = 1 then
        message 'message with message ID ' || @msgid || ' has been
redelivered';
    end if;
    commit;
end
```

ml_qa_getreplytoaddress

Returns the address to which a reply to this message should be sent.

Parameters

Item	Description	Remarks
1	Message ID	VARCHAR(128). You can obtain the message ID from ml_qa_createmessage or ml_qa_getmessage.

Return value

The reply address as VARCHAR(128).

Remarks

You can read this header after a message is received and until a rollback or commit occurs; after that you cannot read it.

See also

- “SQL application setup” on page 56
- “ml_qa_setreplytoaddress” on page 627
- “ml_qa_createmessage” on page 648
- “ml_qa_getmessage” on page 649

Example

In the following example, if the received message has a reply-to address, then a message is sent to the reply-to-address with the content 'message received':

```
begin
    declare @msgid varchar(128);
    declare @rmsgid varchar(128);
    declare @replytoaddr varchar(128);
    set @msgid = ml_qa_getmessage( 'myaddress' );
    set @replytoaddr = ml_qa_getreplytoaddress( @msgid );
    if @replytoaddr is not null then
        set @rmsgid = ml_qa_createmessage();
        call ml_qa_settextcontent( @rmsgid, 'message received' );
        call ml_qa_putmessage( @rmsgid, @replytoaddr );
    end if;
    commit;
end
```

ml_qa_gettimestamp

Returns the creation time of the message.

Parameters

Item	Description	Remarks
1	Message ID	VARCHAR(128). You can obtain the message ID from ml_qa_createmessage or ml_qa_getmessage.

Return value

The message creation time as TIMESTAMP.

Remarks

The Timestamp header field contains the time a message was created. It is a coordinated universal time (UTC). It is not the time the message was actually transmitted, because the actual send may occur later due to transactions or other client-side queuing of messages.

You can read this header after a message is received and until a rollback or commit occurs; after that you cannot read it.

See also

- "SQL application setup" on page 56
- "ml_qa_createmessage" on page 648
- "ml_qa_getmessage" on page 649

Example

In the following example, a message is received and the creation time of the message is output to the database server messages window:

```
begin
    declare @msgid varchar(128);
    declare @ts timestamp;
    set @msgid = ml_qa_getmessage( 'myaddress' );
    set @ts = ml_qa_gettimestamp( @msgid );
    message 'message received with create time: ' || @ts ;
    commit;
end
```

ml_qa_setexpiration

Sets the expiration time for a message.

Parameters

Item	Description	Remarks
1	Message ID	VARCHAR(128). You can obtain the message ID from ml_qa_createmessage or ml_qa_getmessage.
2	Expiration	TIMESTAMP

Remarks

You can read this header after a message is received and until a rollback or commit occurs; after that you cannot read it.

See also

- "SQL application setup" on page 56
- "ml_qa_getexpiration" on page 619
- "ml_qa_createmessage" on page 648
- "ml_qa_getmessage" on page 649

Example

In the following example, a message is created so that if it is not delivered within the next 3 days it expires:

```
begin
    declare @msgid varchar(128);
    set @msgid = ml_qa_createmessage();
    call ml_qa_setexpiration( @msgid, dateadd( day, 3, current timestamp ) );
```

```
        call ml_qa_settextcontent( @msgid, 'time-limited offer' );
        call ml_qa_putmessage( @msgid, 'clientid\queuename' );
        commit;
    end
```

ml_qa_setinreplytoid

Sets the in-reply-to ID of this message.

Parameters

Item	Description	Remarks
1	Message ID	VARCHAR(128). You can obtain the message ID from ml_qa_createmessage or ml_qa_getmessage.
2	in-reply-to ID	VARCHAR(128)

Remarks

An in-reply-to ID is similar to the in-reply-to IDs that are used by email systems to track replies.

Typically you set the in-reply-to ID to be the message ID of the message to which this message is replying, if any.

A client can use the InReplyToID header field to link one message with another. A typical use is to link a response message with its request message.

You cannot alter this header after the message has been sent.

See also

- "SQL application setup" on page 56
- "ml_qa_getinreplytoid" on page 620
- "ml_qa_createmessage" on page 648
- "ml_qa_getmessage" on page 649

Example

In the following example, when a message is received that contains a reply-to-address, a reply message is created and sent containing the message ID in the in-reply-to-id:

```
begin
    declare @msgid varchar(128);
    declare @rmsgid varchar(128);
    declare @replyaddr varchar(128);
    set @msgid = ml_qa_getmessage( 'myaddress' );
    set @replyaddr = ml_qa_getreplyaddress( @msgid );
    if @replyaddr is not null then
        set @rmsgid = ml_qa_createmessage();
        call ml_qa_settextcontent( @rmsgid, 'message received' );
        call ml_qa_setinreplytoid( @rmsgid, @msgid );
        call ml_qa_putmessage( @rmsgid, @replyaddr );
    end if;
```



```
    commit;
end
```

ml_qa_setpriority

Sets the priority of a message.

Parameters

Item	Description	Remarks
1	Message ID	VARCHAR(128). You can obtain the message ID from ml_qa_createmessage or ml_qa_getmessage.
2	Priority	INTEGER

Remarks

The QAnywhere API defines ten levels of priority value, with 0 as the lowest priority and 9 as the highest. Clients should consider priorities 0-4 as gradations of normal priority and priorities 5-9 as gradations of expedited priority.

You cannot alter this header after the message has been sent.

See also

- “SQL application setup” on page 56
- “ml_qa_getpriority” on page 621
- “ml_qa_createmessage” on page 648
- “ml_qa_getmessage” on page 649

Example

The following example sends a high priority message:

```
begin
    declare @msgid varchar(128);
    set @msgid = ml_qa_createmessage();
    call ml_qa_setpriority( @msgid, 9 );
    call ml_qa_settextcontent( @msgid, 'priority content' );
    call ml_qa_putmessage( @msgid, 'clientid\queuename' );
    commit;
end
```

ml_qa_setreplytoaddress

Sets the reply-to address of the message.

Parameters

Item	Description	Remarks
1	Message ID	VARCHAR(128). You can obtain the message ID from ml_qa_createmessage or ml_qa_getmessage.
2	Reply address	VARCHAR(128)

Remarks

You cannot alter this header after the message has been sent.

See also

- "SQL application setup" on page 56
- "ml_qa_getreplytoaddress" on page 623
- "ml_qa_createmessage" on page 648
- "ml_qa_getmessage" on page 649

Example

In the following example, a reply-to-address is added to a message. The recipient of the message can then use that reply-to-address to create a reply.

```
begin
    declare @msgid varchar(128);
    set @msgid = ml_qa_createmessage();
    call ml_qa_setreplytoaddress( @msgid, 'myaddress' );
    call ml_qa_settextcontent( @msgid, 'some content' );
    call ml_qa_putmessage( @msgid, 'clientid\queuename' );
    commit;
end
```

Message properties

You can use the following stored procedures to get and set your custom message properties, or to get predefined message properties.

See also

- "Message properties" on page 658

ml_qa_getbooleanproperty

Returns the specified message property as a SQL BIT data type.

Parameters

Item	Description	Remarks
1	Message ID	VARCHAR(128). You can obtain the message ID from ml_qa_createmessage or ml_qa_getmessage.
2	Property name	VARCHAR(128)

Return value

The property value as BIT.

Remarks

If the message property value is out of range, then a SQL error with SQLSTATE 22003 occurs.

You can read this property after a message is received and until a rollback or commit occurs; after that you cannot read it.

See also

- "SQL application setup" on page 56
- "ml_qa_setbooleanproperty" on page 636
- "ml_qa_createmessage" on page 648
- "ml_qa_getmessage" on page 649
- "Custom message properties" on page 661

Example

In the following example, a message is received and the value of the boolean property mybooleanproperty is output to the database server messages window:

```
begin
    declare @msgid varchar(128);
    declare @prop bit;
    set @msgid = ml_qa_getmessage( 'myaddress' );
    set @prop = ml_qa_getbooleanproperty( @msgid, 'mybooleanproperty' );
    message 'message property mybooleanproperty is set to ' || @prop;
    commit;
end
```

ml_qa_getbyteproperty

Returns the specified message property as a SQL TINYINT data type.

Parameters

Item	Description	Remarks
1	Message ID	VARCHAR(128). You can obtain the message ID from ml_qa_createmessage or ml_qa_getmessage.

Item	Description	Remarks
2	Property name	VARCHAR(128)

Return value

The property value as TINYINT.

Remarks

If the message property value is out of range, then a SQL error with SQLSTATE 22003 occurs.

You can read this property after a message is received and until a rollback or commit occurs; after that you cannot read it.

See also

- "SQL application setup" on page 56
- "ml_qa_setbyteproperty" on page 637
- "ml_qa_createmessage" on page 648
- "ml_qa_getmessage" on page 649
- "Custom message properties" on page 661

Example

In the following example, a message is received and the value of byte property mybyteproperty is output to the database server messages window:

```
begin
    declare @msgid varchar(128);
    declare @prop tinyint;
    set @msgid = ml_qa_getmessage( 'myaddress' );
    set @prop = ml_qa_getbyteproperty( @msgid, 'mybyteproperty' );
    message 'message property mybyteproperty is set to ' || @prop;
    commit;
end
```

ml_qa_getdoubleproperty

Returns the specified message property as a SQL DOUBLE data type.

Parameters

Item	Description	Remarks
1	Message ID	VARCHAR(128). You can obtain the message ID from ml_qa_createmessage or ml_qa_getmessage.
2	Property name	VARCHAR(128)

Return value

The property value as DOUBLE.

Remarks

If the message property value is out of range, then a SQL error with SQLSTATE 22003 occurs.

You can read this property after a message is received and until a rollback or commit occurs; after that you cannot read it.

See also

- "SQL application setup" on page 56
- "ml_qa_setdoubleproperty" on page 638
- "ml_qa_createmessage" on page 648
- "ml_qa_getmessage" on page 649
- "Custom message properties" on page 661

Example

In the following example, a message is received and the value of double property mydoubleproperty is output to the database server messages window:

```
begin
    declare @msgid varchar(128);
    declare @prop double;
    set @msgid = ml_qa_getmessage( 'myaddress' );
    set @prop = ml_qa_getdoubleproperty( @msgid, 'mydoubleproperty' );
    message 'message property mydoubleproperty is set to ' || @prop;
    commit;
end
```

ml_qa_getfloatproperty

Returns the specified message property as a SQL FLOAT data type.

Parameters

Item	Description	Remarks
1	Message ID	VARCHAR(128). You can obtain the message ID from ml_qa_createmessage or ml_qa_getmessage.
2	Property name	VARCHAR(128)

Return value

The property value as FLOAT.

Remarks

If the message property value is out of range, then a SQL error with SQLSTATE 22003 occurs.

You can read this property after a message is received and until a rollback or commit occurs; after that you cannot read it.

See also

- "SQL application setup" on page 56
- "ml_qa_setfloatproperty" on page 639
- "ml_qa_createmessage" on page 648
- "ml_qa_getmessage" on page 649
- "Custom message properties" on page 661

Example

In the following example, a message is received and the value of float property myfloatproperty is output to the database server messages window:

```
begin
    declare @msgid varchar(128);
    declare @prop float;
    set @msgid = ml_qa_getmessage( 'myaddress' );
    set @prop = ml_qa_getfloatproperty( @msgid, 'myfloatproperty' );
    message 'message property myfloatproperty is set to ' || @prop;
    commit;
end
```

ml_qa_getintproperty

Returns the specified message property as a SQL INTEGER data type.

Parameters

Item	Description	Remarks
1	Message ID	VARCHAR(128). You can obtain the message ID from ml_qa_createmessage or ml_qa_getmessage.
2	Property name	VARCHAR(128)

Return value

The property value as INTEGER.

Remarks

If the message property value is out of range, then a SQL error with SQLSTATE 22003 occurs.

You can read this property after a message is received and until a rollback or commit occurs; after that you cannot read it.

See also

- "SQL application setup" on page 56
- "ml_qa_setintproperty" on page 640
- "ml_qa_createmessage" on page 648
- "ml_qa_getmessage" on page 649
- "Custom message properties" on page 661

Example

In the following example, a message is received and the value of integer property myintproperty is output to the database server messages window:

```
begin
    declare @msgid varchar(128);
    declare @prop integer;
    set @msgid = ml_qa_getmessage( 'myaddress' );
    set @prop = ml_qa_getintproperty( @msgid, 'myintproperty' );
    message 'message property myintproperty is set to ' || @prop;
    commit;
end
```

ml_qa_getlongproperty

Returns the specified message property as a SQL BIGINT data type.

Parameters

Item	Description	Remarks
1	Message ID	VARCHAR(128). You can obtain the message ID from ml_qa_createmessage or ml_qa_getmessage.
2	Property name	VARCHAR(128

Return value

The property value as BIGINT.

Remarks

If the message property value is out of range, then a SQL error with SQLSTATE 22003 occurs.

You can read this property after a message is received and until a rollback or commit occurs; after that you cannot read it.

See also

- "SQL application setup" on page 56
- "ml_qa_setlongproperty" on page 640
- "ml_qa_createmessage" on page 648
- "ml_qa_getmessage" on page 649
- "Custom message properties" on page 661

ml_qa_getpropertynames

Retrieves the property names of the specified message.

Parameters

Item	Description	Remarks
1	Message ID	VARCHAR(128). You can obtain the message ID from ml_qa_createmessage or ml_qa_getmessage.

Remarks

This stored procedure opens a result set over the property names of the specified message. The message ID parameter must be that of a message that has been received.

The result set is a single VARCHAR(128) column, where each row contains the name of a message property. QAnywhere reserved property names (those with the prefix "ias_" or "QA") are not returned.

You can read this property after a message is received and until a rollback or commit occurs; after that you cannot read it.

See also

- “SQL application setup” on page 56
- “ml_qa_createmessage” on page 648
- “ml_qa_getmessage” on page 649
- “Custom message properties” on page 661

Example

The following example declares a cursor over the result set of property names for a message that has the message ID msgid. It then gets a message that has the address clientid\queuename; opens a cursor to access the property names of the message; and finally fetches the next property name.

```
begin
 declare prop_name_cursor cursor for
  call ml_qa_getpropertynames( @msgid );
 declare @msgid varchar(128);
 declare @name varchar(128);

 set @msgid = ml_qa_getmessage( 'clientid\queuename' );
 open prop_name_cursor;
 lp: loop
  fetch next prop_name_cursor into name;
  if sqlcode <> 0 then leave lp end if;
  ...
 end loop;
 close prop_name_cursor;
end
```


ml_qa_getshortproperty

Returns the specified message property as a SQL SMALLINT data type.

Parameters

Item	Description	Remarks
1	Message ID	VARCHAR(128). You can obtain the message ID from ml_qa_createmessage or ml_qa_getmessage.
2	Property name	VARCHAR(128)

Return value

The property value as SMALLINT.

Remarks

If the message property value is out of range, then a SQL error with SQLSTATE 22003 occurs.

You can read this property after a message is received and until a rollback or commit occurs; after that you cannot read it.

See also

- "SQL application setup" on page 56
- "ml_qa_setshortproperty" on page 641
- "ml_qa_createmessage" on page 648
- "ml_qa_getmessage" on page 649
- "Custom message properties" on page 661

Example

In the following example, a message is received and the value of the short property myshortproperty is output to the database server messages window:

```
begin
    declare @msgid varchar(128);
    declare @prop smallint;
    set @msgid = ml_qa_getmessage( 'myaddress' );
    set @prop = ml_qa_getshortproperty( @msgid, 'myshortproperty' );
    message 'message property myshortproperty is set to ' || @prop;
    commit;
end
```

ml_qa_getstringproperty

Returns the specified message property as a SQL LONG VARCHAR data type.

Parameters

Item	Description	Remarks
1	Message ID	VARCHAR(128). You can obtain the message ID from ml_qa_createmessage or ml_qa_getmessage.
2	Property name	VARCHAR(128)

Return value

The property value as LONG VARCHAR.

Remarks

If the message property value is out of range, then a SQL error with SQLSTATE 22003 occurs.

You can read this property after a message is received and until a rollback or commit occurs; after that you cannot read it.

See also

- "SQL application setup" on page 56
- "ml_qa_setstringproperty" on page 642
- "ml_qa_createmessage" on page 648
- "ml_qa_getmessage" on page 649
- "Custom message properties" on page 661

Example

In the following example, a message is received and the value of the string property mystringproperty is output to the database server messages window:

```
begin
    declare @msgid varchar(128);
    declare @prop long varchar;
    set @msgid = ml_qa_getmessage( 'myaddress' );
    set @prop = ml_qa_getstringproperty( @msgid, 'mystringproperty' );
    message 'message property mystringproperty is set to ' || @prop;
    commit;
end
```

ml_qa_setbooleanproperty

Sets the specified message property from a SQL BIT data type.

Parameters

Item	Description	Remarks
1	Message ID	VARCHAR(128). You can obtain the message ID from ml_qa_createmessage or ml_qa_getmessage.

Item	Description	Remarks
2	Property name	VARCHAR(128)
3	Property value	BIT

Remarks

You cannot alter this property after the message has been sent.

See also

- "SQL application setup" on page 56
- "ml_qa_getbooleanproperty" on page 628
- "ml_qa_createmessage" on page 648
- "ml_qa_getmessage" on page 649
- "Custom message properties" on page 661

Example

In the following example, a message is created, the boolean properties mybooleanproperty1 and mybooleanproperty2 are set, and the message is sent to the address clientid\queuename:

```
begin
    declare @msgid varchar(128);
    set @msgid = ml_qa_createmessage();
    call ml_qa_setbooleanproperty( @msgid, 'mybooleanproperty1', 0 );
    call ml_qa_setbooleanproperty( @msgid, 'mybooleanproperty2', 1 );
    call ml_qa_putmessage( @msgid, 'clientid\queuename' );
    commit;
end
```

ml_qa_setbyteproperty

Sets the specified message property from a SQL TINYINT data type.

Parameters

Item	Description	Remarks
1	Message ID	VARCHAR(128). You can obtain the message ID from ml_qa_createmessage or ml_qa_getmessage.
2	Property name	VARCHAR(128)
3	Property value	TINYINT

Remarks

You cannot alter this property after the message has been sent.

See also

- “SQL application setup” on page 56
- “ml_qa_getbyteproperty” on page 629
- “ml_qa_createmessage” on page 648
- “ml_qa_getmessage” on page 649
- “Custom message properties” on page 661

Example

In the following example, a message is created, the byte properties mybyteproperty1 and mybyteproperty2 are set, and the message is sent to the address clientid\queuename:

```
begin
    declare @msgid varchar(128);
    set @msgid = ml_qa_createmessage();
    call ml_qa_setbyteproperty( @msgid, 'mybyteproperty1', 0 );
    call ml_qa_setbyteproperty( @msgid, 'mybyteproperty2', 255 );
    call ml_qa_putmessage( @msgid, 'clientid\queuename' );
    commit;
end
```

ml_qa_setdoubleproperty

Sets the specified message property from a SQL DOUBLE data type.

Parameters

Item	Description	Remarks
1	Message ID	VARCHAR(128). You can obtain the message ID from ml_qa_createmessage or ml_qa_getmessage.
2	Property name	VARCHAR(128)
3	Property value	DOUBLE

Remarks

You cannot alter this property after the message has been sent.

See also

- “SQL application setup” on page 56
- “ml_qa_getdoubleproperty” on page 630
- “ml_qa_createmessage” on page 648
- “ml_qa_getmessage” on page 649
- “Custom message properties” on page 661

Example

In the following example, a message is created, the double properties mydoubleproperty1 and mydoubleproperty2 are set, and the message is sent to the address clientid\queuename:


```
begin
    declare @msgid varchar(128);
    set @msgid = ml_qa_createmessage();
    call ml_qa_setdoubleproperty( @msgid, 'mydoubleproperty1', -12.34e-56 );
    call ml_qa_setdoubleproperty( @msgid, 'mydoubleproperty2', 12.34e56 );
    call ml_qa_putmessage( @msgid, 'clientid\queuename' );
    commit;
end
```

ml_qa_setfloatproperty

Sets the specified message property from a SQL FLOAT data type.

Parameters

Item	Description	Remarks
1	Message ID	VARCHAR(128). You can obtain the message ID from ml_qa_createmessage or ml_qa_getmessage.
2	Property name	VARCHAR(128)
3	Property value	FLOAT

Remarks

You cannot alter this property after the message has been sent.

See also

- “SQL application setup” on page 56
- “ml_qa_getfloatproperty” on page 631
- “ml_qa_createmessage” on page 648
- “ml_qa_getmessage” on page 649
- “Custom message properties” on page 661

Example

In the following example, a message is created, the float properties myfloatproperty1 and myfloatproperty2 are set, and the message is sent to the address clientid\queuename:

```
begin
    declare @msgid varchar(128);
    set @msgid = ml_qa_createmessage();
    call ml_qa_setfloatproperty( @msgid, 'myfloatproperty1', -1.3e-5 );
    call ml_qa_setfloatproperty( @msgid, 'myfloatproperty2', 1.3e5 );
    call ml_qa_putmessage( @msgid, 'clientid\queuename' );
    commit;
end
```

ml_qa_setintproperty

Sets the specified message property from a SQL INTEGER data type.

Parameters

Item	Description	Remarks
1	Message ID	VARCHAR(128). You can obtain the message ID from ml_qa_createmessage or ml_qa_getmessage.
2	Property name	VARCHAR(128)
3	Property value	INTEGER

Remarks

You cannot alter this property after the message has been sent.

See also

- “SQL application setup” on page 56
- “ml_qa_getintproperty” on page 632
- “ml_qa_createmessage” on page 648
- “ml_qa_getmessage” on page 649
- “Custom message properties” on page 661

Example

In the following example, a message is created, the integer properties myintproperty1 and myintproperty2 are set, and the message is sent to the address clientid\queuename:

```
begin
    declare @msgid varchar(128);
    set @msgid = ml_qa_createmessage();
    call ml_qa_setintproperty( @msgid, 'myintproperty1', -1234567890 );
    call ml_qa_setintproperty( @msgid, 'myintproperty2',  1234567890 );
    call ml_qa_putmessage( @msgid, 'clientid\queuename' );
    commit;
end
```

ml_qa_setlongproperty

Sets the specified message property from a SQL BIGINT data type.

Parameters

Item	Description	Remarks
1	Message ID	VARCHAR(128). You can obtain the message ID from ml_qa_createmessage or ml_qa_getmessage.

Item	Description	Remarks
2	Property name	VARCHAR(128)
3	Property value	BIGINT

Remarks

You cannot alter this property after the message has been sent.

See also

- "SQL application setup" on page 56
- "ml_qa_getlongproperty" on page 633
- "ml_qa_createmessage" on page 648
- "ml_qa_getmessage" on page 649
- "Custom message properties" on page 661

Example

In the following example, a message is created, the long properties mylongproperty1 and mylongproperty2 are set, and the message is sent to the address clientid\queuename:

```
begin
    declare @msgid varchar(128);
    set @msgid = ml_qa_createmessage();
    call ml_qa_setlongproperty( @msgid, 'mylongproperty1',
-12345678900987654321 );
    call ml_qa_setlongproperty( @msgid, 'mylongproperty2',
12345678900987654321 );
    call ml_qa_putmessage( @msgid, 'clientid\queuename' );
    commit;
end
```

ml_qa_setshortproperty

Sets the specified message property from a SQL SMALLINT data type.

Parameters

Item	Description	Remarks
1	Message ID	VARCHAR(128). You can obtain the message ID from ml_qa_createmessage or ml_qa_getmessage.
2	Property name	VARCHAR(128)
3	Property value	SMALLINT

Remarks

You cannot alter this property after the message has been sent.

See also

- “SQL application setup” on page 56
- “ml_qa_getshortproperty” on page 635
- “ml_qa_createmessage” on page 648
- “ml_qa_getmessage” on page 649
- “Custom message properties” on page 661

Example

In the following example, a message is created, the short properties myshortproperty1 and myshortproperty2 are set, and the message is sent to the address clientid\queuename:

```
begin
    declare @msgid varchar(128);
    set @msgid = ml_qa_createmessage();
    call ml_qa_setshortproperty( @msgid, 'myshortproperty1', -12345 );
    call ml_qa_setshortproperty( @msgid, 'myshortproperty2',  12345 );
    call ml_qa_putmessage( @msgid, 'clientid\queuename' );
    commit;
end
```

ml_qa_setstringproperty

Sets the specified message property from a SQL LONG VARCHAR data type.

Parameters

Item	Description	Remarks
1	Message ID	VARCHAR(128). You can obtain the message ID from ml_qa_createmessage or ml_qa_getmessage.
2	Property name	VARCHAR(128)
3	Property value	LONG VARCHAR

Remarks

You cannot alter this property after the message has been sent.

See also

- “SQL application setup” on page 56
- “ml_qa_getstringproperty” on page 635
- “ml_qa_createmessage” on page 648
- “ml_qa_getmessage” on page 649
- “Custom message properties” on page 661

Example

In the following example, a message is created, the string properties mystringproperty1 and mystringproperty2 are set, and the message is sent to the address clientid\queuename:


```
begin
    declare @msgid varchar(128);
    set @msgid = ml_qa_createmessage();
    call ml_qa_setstringproperty( @msgid, 'mystringproperty1', 'c:\\temp' );
    call ml_qa_setstringproperty( @msgid, 'mystringproperty2',  'first line
\nsecond line' );
    call ml_qa_putmessage( @msgid, 'clientid\queuename' );
    commit;
end
```

Message content

You can use the following stored procedures to get and set message content.

ml_qa_getbinarycontent

Returns the message content of a binary message.

Parameters

Item	Description	Remarks
1	Message ID	VARCHAR(128). You can obtain the message ID from ml_qa_createmessage or ml_qa_getmessage.

Return value

The message content as LONG BINARY.

If the message has text content rather than binary content, this stored procedure returns null.

You can read this content after a message is received and until a rollback or commit occurs; after that you cannot read it.

See also

- “SQL application setup” on page 56
- “ml_qa_setbinarycontent” on page 645
- “ml_qa_createmessage” on page 648
- “ml_qa_getmessage” on page 649
- “ml_qa_getcontentclass” on page 644

Example

In the following example, a message's encrypted content is decrypted and output to the database server messages window:

```
begin
    declare @msgid varchar(128);
    declare @content long binary;
    declare @plaintext long varchar;
    set @msgid = ml_qa_getmessage( 'myaddress' );
    set @content = ml_qa_getbinarycontent( @msgid );
```

```
        set @plaintext = decrypt( @content, 'mykey' );
        message 'message content decrypted: ' || @plaintext;
        commit;
    end
```

ml_qa_getcontentclass

Returns the message type (text or binary).

Parameters

Item	Description	Remarks
1	Message ID	VARCHAR(128). You can obtain the message ID from ml_qa_createmessage or ml_qa_getmessage.

Return value

The content class as INTEGER.

The return value can be:

- **1** indicates that the message content is binary and should be read using the stored procedure ml_qa_getbinarycontent.
- **2** indicates that the message content is text and should be read using the stored procedure ml_qa_gettextcontent.

Remarks

You can read this content after a message is received and until a rollback or commit occurs; after that you cannot read it.

See also

- “SQL application setup” on page 56
- “ml_qa_createmessage” on page 648
- “ml_qa_getmessage” on page 649
- “ml_qa_getbinarycontent” on page 643
- “ml_qa_gettextcontent” on page 645

Example

In the following example, a message is received and the content is output to the database server messages window:

```
begin
    declare @msgid varchar(128);
    declare @contentclass integer;
    set @msgid = ml_qa_getmessage( 'myaddress' );
    set @contentclass = ml_qa_getcontentclass( @msgid );
    if @contentclass = 1 then
        message 'message  binary is ' || ml_qa_getbinarycontent( @msgid );
```



```
    elseif @contentclass = 2 then
        message 'message  text is ' || ml_qa_gettextcontent( @msgid );
    end if;
    commit;
end
```

ml_qa_gettextcontent

Returns the message content of a text message.

Parameters

Item	Description	Remarks
1	Message ID	VARCHAR(128). You can obtain the message ID from ml_qa_createmessage or ml_qa_getmessage.

Return value

The text content as LONG VARCHAR.

If the message has binary content rather than text content, this stored procedure returns null.

Remarks

You can read this content after a message is received and until a rollback or commit occurs; after that you cannot read it.

See also

- "SQL application setup" on page 56
- "ml_qa_settextcontent" on page 646
- "ml_qa_createmessage" on page 648
- "ml_qa_getmessage" on page 649
- "ml_qa_getcontentclass" on page 644

Example

In the following example, the content of a message is output to the database server messages window:

```
begin
    declare @msgid varchar(128);
    declare @content long binary;
    set @msgid = ml_qa_getmessage( 'myaddress' );
    set @content = ml_qa_gettextcontent( @msgid );
    message 'message content: ' || @content ;
    commit;
end
```

ml_qa_setbinarycontent

Sets the binary content of the message.

Parameters

Item	Description	Remarks
1	Message ID	VARCHAR(128). You can obtain the message ID from ml_qa_createmessage or ml_qa_getmessage.
2	Content	LONG BINARY

You cannot alter this content after the message has been sent.

See also

- "SQL application setup" on page 56
- "ml_qa_getbinarycontent" on page 643
- "ml_qa_createmessage" on page 648
- "ml_qa_getmessage" on page 649

Example

In the following example, a message is created with encrypted content and sent:

```
begin
    declare @msgid varchar(128);
    set @msgid = ml_qa_createmessage();
    call ml_qa_setbinarycontent( @msgid, encrypt( 'my secret message',
'mykey' ) );
    call ml_qa_putmessage( @msgid, 'clientid\queuename' );
    commit;
end
```

ml_qa_settextcontent

Sets the text content of the message.

Parameters

Item	Description	Remarks
1	Message ID	VARCHAR(128). You can obtain the message ID from ml_qa_createmessage or ml_qa_getmessage.
2	Content	LONG VARCHAR

Remarks

You cannot alter this content after the message has been sent.

See also

- "SQL application setup" on page 56
- "ml_qa_gettextcontent" on page 645
- "ml_qa_createmessage" on page 648
- "ml_qa_getmessage" on page 649

Example

In the following example, a message is created and then set with the given content:

```
begin
    declare @msgid varchar(128);
    set @msgid = ml_qa_createmessage();
    call ml_qa_settextcontent( @msgid, 'my simple message' );
    call ml_qa_putmessage( @msgid, 'clientid\queuename' );
    commit;
end
```

Message store properties

You can use the following stored procedures to get and set properties for client message stores.

For more information about message store properties, see "Client message store properties" on page 26.

ml_qa_getstoreproperty

Returns a client message store property.

Parameters

Item	Description	Remarks
1	Property name	VARCHAR(128)

Return value

The property value as LONG VARCHAR.

Remarks

Client message store properties are readable from every connection to this client message store.

See also

- "SQL application setup" on page 56
- "ml_qa_setstoreproperty" on page 648

Example

The following example gets the current synchronization policy of this message store and outputs it to the database server messages window:

```
begin
    declare @policy varchar(128);
    set @policy = ml_qa_getstoreproperty( 'policy' );
    message 'the current policy for synchronizing this message store is ' ||
@policy;
end
```

ml_qa_setstoreproperty

Sets a client message store property.

Parameters

Item	Description	Remarks
1	Property name	VARCHAR(128)
2	Property value	SMALLINT

Remarks

Client message store properties are readable from every connection to this client message store. The values are synchronized up to the server, as well, where they can be used in transmission rules.

See also

- “SQL application setup” on page 56
- “ml_qa_getstoreproperty” on page 647

Example

The following example sets the synchronization policy to automatic for the message store:

```
begin
    call ml_qa_setstoreproperty( 'policy', 'automatic' );
    commit;
end
```

Message management

You can use the following stored procedures to manage your QAnywhere client transactions.

ml_qa_createmessage

Returns the message ID of a new message.

Return value

The message ID of the new message.

Remarks

Use this stored procedure to create a message. Once created, you can associate content, properties, and headers with this message and then send the message.

You can associate content, properties, and headers using any of the QAnywhere stored procedures starting with ml_qa_set. For example, use ml_qa_setbinarycontent or ml_qa_settextcontent to create a binary or text message.

See also

- "SQL application setup" on page 56
- "Message headers" on page 618
- "Message properties" on page 628
- "Message content" on page 643

Example

The following example creates a message, sets the message content, and sends the message to the address clientid\queuename:

```
begin
    declare @msgid varchar(128);
    set @msgid = ml_qa_createmessage();
    call ml_qa_settextcontent( @msgid, 'some content' );
    call ml_qa_putmessage( @msgid, 'clientid\queuename' );
    commit;
end
```

ml_qa_getmessage

Returns the message ID of the next message that is queued for the given address, blocking until one is queued.

Parameters

Item	Description	Remarks
1	Address	VARCHAR(128)

Return value

The message ID as VARCHAR(128).

Returns null if there is no queued message for this address.

Remarks

Use this stored procedure to check synchronously whether there is a message waiting for the specified QAnywhere message address. Use the MobiLink Listener if you want a SQL procedure to be called asynchronously when a message is available for a specified QAnywhere address.

This stored procedure blocks until a message is queued.

For information about avoiding blocking, see “ml_qa_getmessagenowait” on page 650 or “ml_qa_getmessagetimeout” on page 651.

The message corresponding to the returned message ID is not considered to be received until the current transaction is committed. Once the receive is committed, the message cannot be received again by this or any other QAnywhere API. Similarly, a rollback of the current transaction means that the message is not received, so subsequent calls to ml_qa_getmessage may return the same message ID.

The properties and content of the received message can be read by the various ml_qa_get stored procedures until a commit or rollback is executed on the current transaction. Once a commit or rollback is executed on the current transaction, the message data is no longer readable. Before committing, you should store any data you need from the message as tabular data or in SQL variables.

See also

- “SQL application setup” on page 56
- “ml_qa_getmessagenowait” on page 650
- “ml_qa_getmessagetimeout” on page 651
- “Message headers” on page 618
- “Message properties” on page 628
- “Message content” on page 643

Example

The following example displays the content of all messages sent to the address myaddress:

```
begin
    declare @msgid varchar(128);
    loop
        set @msgid = ml_qa_getmessage( 'myaddress' );
        message 'a message with content ' || ml_qa_gettextcontent( @msgid )
|| ' has been received';
        commit;
    end loop;
end
```

ml_qa_getmessagenowait

Returns the message ID of the next message that is currently queued for the given address.

Parameters

Item	Description	Remarks
1	Address	VARCHAR(128)

Return value

The message ID as VARCHAR(128).

Returns the message ID of the next message that is queued for the given address. Returns null if there is no queued message for this address.

Remarks

Use this stored procedure to check synchronously whether there is a message waiting for the specified QAnywhere message address. Use the MobiLink Listener if you want a SQL procedure to be called asynchronously when a message is available for a specified QAnywhere address.

For information about blocking until a message is available, see "ml_qa_getmessage" on page 649 and "ml_qa_getmessagetimeout" on page 651.

The message corresponding to the returned message is not considered to be received until the current transaction is committed. Once the receive is committed, the message cannot be received again by this or any other QAnywhere API. Similarly, a rollback of the current transaction means that the message is not received, so subsequent calls to ml_qa_getmessagenowait may return the same message ID.

The properties and content of the received message can be read by the various ml_qa_get stored procedures until a commit or rollback is executed on the current transaction. Once a commit or rollback is executed on the current transaction, the message data is no longer readable. Before committing, you should store any data you need from the message as tabular data or in SQL variables.

See also

- "SQL application setup" on page 56
- "QAnywhere message addresses" on page 57
- "Listeners" [*MobiLink - Server-Initiated Synchronization*]
- "ml_qa_getmessagetimeout" on page 651
- "Message headers" on page 618
- "Message properties" on page 628
- "Message content" on page 643

Example

The following example displays the content of all messages that are queued at the address myaddress until all such messages are read (it is generally more efficient to commit after the last message has been read, rather than after each message is read):

```
begin
    declare @msgid varchar(128);
    loop
        set @msgid = ml_qa_getmessagenowait( 'myaddress' );
        if @msgid is null then leave end if;
        message 'a message with content ' || ml_qa_gettextcontent( @msgid )
|| ' has been received';
    end loop;
    commit;
end
```

ml_qa_getmessagetimeout

Waits for the specified timeout period to return the message ID of the next message that is queued for the given address.

Parameters

Item	Description	Remarks
1	Address	VARCHAR(128)
2	Timeout in milliseconds	INTEGER

Return value

The message ID as VARCHAR(128).

Returns null if there is no queued message for this address within the timeout period.

Remarks

Use this stored procedure to check synchronously whether there is a message waiting for the specified QAnywhere message address. Use the MobiLink Listener if you want a SQL procedure to be called asynchronously when a message is available for a specified QAnywhere address.

The message corresponding to the returned message is not considered to be received until the current transaction is committed. Once the receive is committed, the message cannot be received again by this or any other QAnywhere API. Similarly, a rollback of the current transaction means that the message is not received, so subsequent calls to ml_qa_getmessage may return the same message ID.

The properties and content of the received message can be read by the various ml_qa_get stored procedures until a commit or rollback is executed on the current transaction. Once a commit or rollback is executed on the current transaction, the message data is no longer readable. Before committing, you should store any data you need from the message as tabular data or in SQL variables.

See also

- “SQL application setup” on page 56
- “ml_qa_getmessage” on page 649
- “ml_qa_getmessagenowait” on page 650

Example

The following example outputs the content of all messages sent to the address myaddress to the database server messages window, and updates the database server messages window every 10 seconds if no message has been received:

```
begin
    declare @msgid varchar(128);
    loop
        set @msgid = ml_qa_getmessagetimeout( 'myaddress', 10000 );
        if @msgid is null then
            message 'waiting for a message...';
        else
            message 'a message with content ' ||
ml_qa_gettextcontent( @msgid ) || ' has been received';
        commit;
        end if;
    end loop;
end
```


ml_qa_grant_messaging_permissions

Grants permission to other users to use QAnywhere stored procedures.

Parameters

Item	Description	Remarks
1	Database user ID	VARCHAR(128)

Remarks

Only users with DBA privilege automatically have permission to execute the QAnywhere stored procedures. Other users must be granted permission by having a user with DBA privileges run this stored procedure.

This procedure adds the user to a group called ml_qa_message_group and gives them execute permissions on all QAnywhere stored procedures.

See also

- "SQL application setup" on page 56

Example

For example, to grant messaging permissions to a user with the database ID user1, execute the following code:

```
call dbo.ml_qa_grant_messaging_permissions( 'user1' )
```

ml_qa_listener_queue

Create a stored procedure named **ml_qa_listener_*queue*** (where *queue* is the name of a message queue) to receive messages asynchronously.

Parameters

Item	Description	Remarks
1	Message ID	VARCHAR(128). You can obtain the message ID from the QAnywhere Listener.

Remarks

Note
This procedure is different from all the other QAnywhere stored procedures in that the stored procedure is not provided. If you create a stored procedure named **ml_qa_listener_*queue***, where *queue* is a message queue, then it is used by QAnywhere.

Although messages can be received synchronously on a connection, it is often convenient to receive messages asynchronously. You can create a stored procedure that is called when a message has been queued on a particular address. The name of this procedure must be **ml_qa_listener_*queue***, where *queue* is the message queue. When this procedure exists, the procedure is called whenever a message is queued on the given address.

This procedure is called from a separate connection. As long as a SQL error does not occur while this procedure is executing, the message is automatically acknowledged and committed.

Note
Do not commit or rollback within this procedure.

The queue name is part of the QAnywhere address.

See also

- "SQL application setup" on page 56
- "QAnywhere message addresses" on page 57
- "Receiving messages asynchronously" on page 69
- "Receiving messages synchronously" on page 68
- "ml_qa_createmessage" on page 648
- "ml_qa_getmessage" on page 649

Example

The following example creates a procedure that is called whenever a message is queued on the address named executesql. In this example, the procedure assumes that the content of the message is a SQL statement that it can execute against the current database.

```
CREATE PROCEDURE ml_qa_listener_executesql(IN @msgid VARCHAR(128))
begin
 DECLARE @execstr LONG VARCHAR;
 SET @execstr = ml_qa_gettextcontent( @msgid );
 EXECUTE IMMEDIATE @execstr;
end
```

ml_qa_putmessage

Sends a message.

Parameters

Item	Description	Remarks
1	Message ID	VARCHAR(128). You can obtain the message ID from ml_qa_createmessage or ml_qa_getmessage.
2	Address	VARCHAR(128)

Remarks

The message ID you specify must have been previously created using ml_qa_createmessage. Only content, properties and headers associated with the message ID before the call to ml_qa_putmessage are sent with the message. Any added after the ml_qa_putmessage are ignored.

A commit is required before the message is actually queued for sending.

See also

- "SQL application setup" on page 56
- "ml_qa_createmessage" on page 648
- "ml_qa_getmessage" on page 649

Example

In the following example, a message is created with the content 'a simple message' and sent to the address clientid\queuename:

```
begin
    declare @msgid varchar(128);
    set @msgid = ml_qa_createmessage();
    call ml_qa_settextcontent( @msgid, 'a simple message' );
    call ml_qa_putmessage( @msgid, 'clientid\queuename' );
    commit;
end
```

ml_qa_triggersendreceive

Triggers a synchronization of messages with the MobiLink server.

Remarks

Normally, message synchronization is handled by the QAnywhere Agent. However, if the synchronization policy is on demand, then it is the application's responsibility to trigger the synchronization of messages. You can do so using this stored procedure. The trigger does not take effect until the current transaction is committed.

See also

- "SQL application setup" on page 56

Example

In the following example, a message is sent and the transmission of the message is immediately initiated:

```
begin
    declare @msgid varchar(128);
    set @msgid = ml_qa_createmessage();
    call ml_qa_settextcontent( @msgid, 'my simple message' );
    call ml_qa_putmessage( @msgid, 'clientid\queuename' );
    call ml_qa_triggersendreceive();
    commit;
end
```

Message headers and properties

Message headers

All QAnywhere messages support the same set of header fields. Header fields contain values that are used by both clients and providers to identify and route messages.

The following message headers are predefined. How you use them depends on the type of client application you have.

- **Message ID** Read-only. The message ID of the new message. This header has a value only after the message is sent. See:
  - "QAMessage.MessageID property [QAnywhere .NET]" on page 288
  - "QAMessage.getMessageID method [QAnywhere C++]" on page 438 and "QAMessage.setMessageID method [QAnywhere C++]" on page 448
  - "QAMessage.getMessageID method [QAnywhere Java]" on page 557
  - SQL API: "ml_qa_createmessage" on page 648 and "ml_qa_getmessage" on page 649
- **Message creation timestamp** Read-only. The Timestamp header field contains the time a message was created. It is a coordinated universal time (UTC). It is not the time the message was actually transmitted, because the actual send may occur later due to transactions or other client-side queuing of messages. You can read this header after a message is received and until a rollback or commit occurs; after that you cannot read it. See:
  - "QAMessage.Timestamp property [QAnywhere .NET]" on page 290
  - "QAMessage.getTimestamp method [QAnywhere C++]" on page 442 and "QAMessage.setTimestamp method [QAnywhere C++]" on page 450
  - "QAMessage.getTimestamp method [QAnywhere Java]" on page 560
  - SQL API: "ml_qa_gettimestamp" on page 624
- **Reply-to address** Read-write. The reply address as VARCHAR(128) or null if it does not exist. You can read this header after a message is received and until a rollback or commit occurs; after that you cannot read it. See:
  - "QAMessage.ReplyToAddress property [QAnywhere .NET]" on page 290
  - "QAMessage.getReplyToAddress method [QAnywhere C++]" on page 440 and "QAMessage.setReplyToAddress method [QAnywhere C++]" on page 449
  - "QAMessage.getReplyToAddress method [QAnywhere Java]" on page 559 and "QAMessage.setReplyToAddress method [QAnywhere Java]" on page 565
  - SQL API: "ml_qa_getreplytoaddress" on page 623 and "ml_qa_setreplytoaddress" on page 627

- **Message address** Read-only. The QAnywhere message address as VARCHAR(128). QAnywhere message addresses take the form *id\queue-name*. You can read this header after a message is received and until a rollback or commit occurs; after that you cannot read it. See:
  - "QAMessage.Address property [QAnywhere .NET]" on page 287
  - "QAMessage.getAddress method [QAnywhere C++]" on page 433
  - "QAMessage.getAddress method [QAnywhere Java]" on page 552
  - SQL API: "ml_qa_getaddress" on page 618

- **Redelivered state of message** Read-only. The redelivered value as BIT. A value of 1 indicates that the message is being redelivered; 0 indicates that it is not being redelivered.

  A message may be redelivered if it was previously received but not acknowledged. For example, the message was received but the application receiving the message did not complete processing the message content before it crashed. In these cases, QAnywhere marks the message as redelivered to alert the receiver that the message might be partly processed.

  For example, assume that the receipt of a message occurs in three steps:

  1. An application using a non-transactional QAnywhere manager receives the message.
  2. The application writes the message content and message ID to a database table called T1, and commits the change.
  3. The application acknowledges the message.

  If the application fails between steps 1 and 2 or between steps 2 and 3, the message is redelivered when the application restarts.

  If the failure occurs between steps 1 and 2, you should process the redelivered message by running steps 2 and 3. If the failure occurs between steps 2 and 3, then the message is already processed and you only need to acknowledge it.

  To determine what happened when the application fails, you can have the application call ml_qa_getredelivered to check if the message has been previously redelivered. Only messages that are redelivered need to be looked up in table T1. This is more efficient than having the application access the received message's message ID to check whether the message is in the table T1, because application failures are rare.

  You can read this header after a message is received and until a rollback or commit occurs; after that you cannot read it.

  See:

  - "QAMessage.Redelivered property [QAnywhere .NET]" on page 289
  - "QAMessage.getRedelivered method [QAnywhere C++]" on page 439 and "QAMessage.setRedelivered method [QAnywhere C++]" on page 449
  - "QAMessage.getRedelivered method [QAnywhere Java]" on page 559
  - SQL API: "ml_qa_getredelivered" on page 622

- **Expiration of message** Read-only except in the SQL API, where it is read-write. The expiration time as TIMESTAMP. Returns null if there is no expiration. A message expires if it is not received by the intended recipient in the specified time. The message may then be deleted using default QAnywhere delete rules. You can read this header after a message is received and until a rollback or commit occurs; after that you cannot read it. See:
  - "QAMessage.Expiration property [QAnywhere .NET]" on page 288
  - "QAMessage.getExpiration method [QAnywhere C++]" on page 435
  - "QAMessage.getExpiration method [QAnywhere Java]" on page 554
  - SQL API: "ml_qa_getexpiration" on page 619 and "ml_qa_setexpiration" on page 625
- **Priority of message** Read-write. The QAnywhere API defines ten levels of priority value, with 0 as the lowest priority and 9 as the highest. Clients should consider priorities 0-4 as gradations of normal priority and priorities 5-9 as gradations of expedited priority. You can read this header after a message is received and until a rollback or commit occurs; after that you cannot read it. See:
  - "QAMessage.Priority property [QAnywhere .NET]" on page 289
  - "QAMessage.getPriority method [QAnywhere C++]" on page 438
  - "QAMessage.getPriority method [QAnywhere Java]" on page 557
  - SQL API: "ml_qa_getpriority" on page 621
- **Message ID of a message for which this message is a reply** Read-write. The in-reply-to ID as VARCHAR(128). A client can use the InReplyToID header field to link one message with another. A typical use is to link a response message with its request message. The in-reply-to ID is the ID of the message that this message is replying to. You can read this header after a message is received and until a rollback or commit occurs; after that you cannot read it. See:
  - "QAMessage.InReplyToID property [QAnywhere .NET]" on page 288
  - "QAMessage.getInReplyToID method [QAnywhere C++]" on page 436
  - "QAMessage.getInReplyToID method [QAnywhere Java]" on page 555
  - SQL API: "ml_qa_getinreplytoid" on page 620

Some message headers can be used in transmission rules. See "Variables defined by the rule engine" on page 735.

See also

- "QAMessage interface [QAnywhere .NET]" on page 272
- "QAMessage class [QAnywhere C++]" on page 429
- "QAMessage interface [QAnywhere Java]" on page 550
- SQL API: "Message headers" on page 618

Message properties

Each message contains a built-in facility for supporting application-defined property values. These message properties allow you to implement application-defined message filtering.

Message properties are name-value pairs that you can optionally insert into messages to provide structure. For example, in the .NET API the predefined message property ias_Originator, identified by the constant

MessageProperties.ORIGINATOR, provides the message store ID that sent the message. Message properties can be used in transmission rules to determine the suitability of a message for transmission.

There are two types of message property:

- **Predefined message properties** These message properties are always prefixed with ias_ or IAS_.
- **Custom message properties** These are message properties that you defined. You cannot prefix them with ias_ or IAS_.

In either case, you access message store properties using get and set methods and pass the name of the predefined or custom property as the first parameter.

See also

- "Message property management" on page 661

Predefined message properties

Some message properties have been predefined for your convenience. Predefined properties can be read but should not be set. The predefined message properties are:

- **ias_Adapters** For network status notification messages, a list of network adapters that can be used to connect to the MobiLink server. The list is a string and is delimited by a vertical bar.
- **ias_DeliveryCount** Int. The number of attempts that have been made so far to deliver the message.
- **ias_MessageType** Int. Indicates the type of the message. The message types can be:

Value	Message type	Description
0	REGULAR	If a message does not have the ias_MessageType property set, it is a regular message.
13	PUSH_NOTIFICATION	When a push notification is received from the server, a message of type PUSH_NOTIFICATION is sent to the system queue. See "Notifications of push notification" on page 60.
14	NETWORK_STATUS_NOTIFICATION	When there is a change in network status, a message of this type is sent to the system queue. See "Network status notifications" on page 59.

- **ias_RASNames** String. For network status notification messages, a list of RAS entry names that can be used to connect to the MobiLink server. The list is delimited by a vertical bar.
- **ias_NetworkStatus** Int. For network status notification messages, the state of the network connection. The value is 1 if connected, 0 otherwise.

- **ias_Originator** String. The message store ID of the originator of the message.
- **ias_Status** Int. The current status of the message. This property is not supported in the SQL API. The values can be:

Status Code	Description
1	Pending - The message has been sent but not received.
10	Receiving - The message is in the process of being received, or it was received but not acknowledged.
20	Final - The message has achieved a final state.
30	Expired - The message was not received before its expiration time has passed.
40	Cancelled - The message has been canceled.
50	Unreceivable - The message is either malformed, or there were too many failed attempts to deliver it.
60	Received - The message has been received and acknowledged.

  There are constants for the status values. See:

  - "StatusCodes enumeration [QAnywhere .NET]" on page 306
  - "StatusCodes class [QAnywhere C++]" on page 464
  - "StatusCodes interface [QAnywhere Java]" on page 581

- **ias_StatusTime** The time at which the message became its current status. It is in units that are natural for the platform. It is a local time. In the C++ API, for Windows and PocketPC platforms, the timestamp is the SYSTEMTIME, converted to a FILETIME, which is copied to a qa_long value. This property is not supported in the SQL API.

API	This property returns...
.NET	DateTime
C++	string
Java	java.util.Date object

Message property constants

The QAnywhere APIs for .NET, C++, and Java provide constants for specifying message properties. See:

- "MessageProperties class [QAnywhere .NET]" on page 181
- "MessageProperties class [QAnywhere C++]" on page 356
- "MessageProperties interface [QAnywhere Java]" on page 469

Custom message properties

QAnywhere allows you to define message properties using the C++, Java, or .NET APIs. Custom message properties allow you to create name-value pairs that you associate with an object. For example:

```
msg.SetStringProperty("Product", "widget");
msg.SetFloatProperty("Price",1.00);
msg.SetIntProperty("Quantity",10);
```

Message property names are case insensitive. You can use a sequence of letters, digits and underscores, but the first character must be a letter. The following names are reserved and may not be used as message property names:

- NULL
- TRUE
- FALSE
- NOT
- AND
- OR
- BETWEEN
- LIKE
- IN
- IS
- ESCAPE
- Any name beginning with **ias_**

Message property management

The following QAMessage methods can be used to manage message properties.

Note
You can get and set custom properties, but should only get predefined properties.

.NET methods to manage message properties

- Object GetProperty(String name)
- void SetProperty(String name, Object value)
- boolean GetBooleanProperty(String name)
- void SetBooleanProperty(String name, boolean value)
- byte GetByteProperty(String name)
- void SetByteProperty(String name, byte value)
- short GetShortProperty(String name)
- void SetShortProperty(String name, short value)
- int GetIntProperty(String name)
- void SetIntProperty(String name, int value)
- long GetLongProperty(String name)
- void SetLongProperty(String name, long value)
- float GetFloatProperty(String name)
- void SetFloatProperty(String name, float value)
- double GetDoubleProperty(String name)
- void SetDoubleProperty(String name, double value)
- String GetStringProperty(String name)
- void SetStringProperty(String name, String value)
- IEnumerator GetPropertyNames()
- void ClearProperties()
- PropertyType GetPropertyType(string propName)
- bool PropertyExists(string propName)

See "QAMessage interface [QAnywhere .NET]" on page 272.

C++ methods to manage message properties

- qa_bool getBooleanProperty(qa_const_string name, qa_bool * value)
- qa_bool setBooleanProperty(qa_const_string name, qa_bool value)
- qa_bool getByteProperty(qa_const_string name, qa_byte * value)
- qa_bool setByteProperty(qa_const_string name, qa_byte value)
- qa_bool getShortProperty(qa_const_string name, qa_short * value)
- qa_bool setShortProperty(qa_const_string name, qa_short value)
- qa_bool getIntProperty(qa_const_string name, qa_int * value)
- qa_bool setIntProperty(qa_const_string name, qa_int value)
- qa_bool getLongProperty(qa_const_string name, qa_long * value)
- qa_bool setLongProperty(qa_const_string name, qa_long value)
- qa_bool getFloatProperty(qa_const_string name, qa_float * value)
- qa_bool setFloatProperty(qa_const_string name, qa_float value)
- qa_bool getDoubleProperty(qa_const_string name, qa_double * value)
- qa_bool setDoubleProperty(qa_const_string name, qa_double value)
- qa_int getStringProperty(qa_const_string name, qa_string value, qa_int len)
- qa_bool setStringProperty(qa_const_string name, qa_const_string value)
- void QAMessage::clearProperties()
- qa_short QAMessage::getPropertyType(qa_const_string name)
- qa_bool QAMessage::propertyExists(qa_const_string name)

See "QAMessage class [QAnywhere C++]" on page 429.

Java methods to manage message properties

- void clearProperties()
- boolean getBooleanProperty(String name)
- void setBooleanProperty(String name, boolean value)
- byte getByteProperty(String name)
- void setByteProperty(String name, byte value)
- double getDoubleProperty(String name)
- void setDoubleProperty(String name, double value)
- java.util.Date getExpiration() void setFloatProperty(String name, float value)
- float getFloatProperty(String name)
- int getIntProperty(String name)
- void setIntProperty(String name, int value)
- long getLongProperty(String name)
- void setLongProperty(String name, long value)
- Object getProperty(String name)
- void setProperty(String name, Object value)
- java.util.Enumeration getPropertyNames()
- short getPropertyType(String name)
- short getShortProperty(String name)
- void setShortProperty(String name, short value)
- String getStringProperty(String name)
- void setStringProperty(String name, String value)
- boolean propertyExists(String name)

See "QAMessage interface [QAnywhere Java]" on page 550.

SQL stored procedures to manage message properties

- ml_qa_getbooleanproperty
- ml_qa_getbyteproperty
- ml_qa_getdoubleproperty
- ml_qa_getfloatproperty
- ml_qa_getintproperty
- ml_qa_getlongproperty
- ml_qa_getpropertynames
- ml_qa_getshortproperty
- ml_qa_getstringproperty
- ml_qa_setbooleanproperty
- ml_qa_setbyteproperty
- ml_qa_setdoubleproperty
- ml_qa_setfloatproperty
- ml_qa_setfloatproperty
- ml_qa_setintproperty
- ml_qa_setlongproperty
- ml_qa_setshortproperty
- ml_qa_setstringproperty

See "Message properties" on page 628.

Example

```
// C++ example.
QAManagerFactory factory;
QAManager * mgr = factory->createQAManager();
mgr->open(AcknowledgementMode::EXPLICIT_ACKNOWLEDGEMENT);
QAMessage * msg = mgr->createTextMessage();
msg->setStringProperty( "tm_Subject", "Some message subject." );
mgr->putMessage( "myqueue", msg );

// C# example.
QAManager mgr = QAManagerFactory.Instance.CreateQAManager();
mgr.Open(AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT);
QAMessage msg = mgr.CreateTextMessage();
msg.SetStringProperty( "tm_Subject", "Some message subject." );
mgr.PutMessage( "myqueue", msg );

// Java example
QAManager mgr = QAManagerFactory.getInstance().createQAManager();
mgr.open(AcknowledgementMode.EXPLICIT_ACKNOWLEDGEMENT);
QAMessage msg = mgr.createTextMessage();
msg.setStringProperty("tm_Subject", "Some message subject.");
mgr.putMessage("myqueue", msg);

-- SQL example
begin
    DECLARE @msgid VARCHAR(128);
    SET @msgid = ml_qa_createmessage();
    CALL ml_qa_setfloatproperty( @msgid, 'myfloatproperty1', -1.3e-5 );
    CALL ml_qa_setfloatproperty( @msgid, 'myfloatproperty2', 1.3e5 );
    CALL ml_qa_putmessage( @msgid, 'clientid\queuename' );
    COMMIT;
end
```

Server management request reference

Server management request parent tags

Condition tag

Use the following condition subtags to filter the messages to include in the MessageDetailsRequest. You can specify as many of these tags as you want in the <condition> tag. If you use more than one of the same tag, then the values given are logically ORed together, whereas if you use two different tags, the values are logically ANDed together.

<condition> subtags	Description
<address>	Selects messages that are addressed to the specified address.
<archived>	Returns the details of messages in the archive message store.
<customRule>	Selects messages based on rules.

<condition> subtags	Description
<kind>	Filters either binary or text messages. For example, <kind>*text*</kind> filters text messages, and <kind>*binary*</kind> filters binary messages.
<messageId>	Selects the message with a particular message ID.
<originator>	Selects messages that originated from the specified client.
<priority>	Selects messages that currently have the priority specified.
<property>	Selects messages that have the specified message property. To check a property name and value, use the syntax <property>*property-name=property-value*</property>. To check the existence of a property, use the format <property>*property-name*</property>.
<status>	Selects messages that currently have the status specified.

CustomRule tag

To construct more complex condition statements, use the <customRule> tag as a subtag to the <condition> tag (and other tags). This tag takes as its data a server rule similar to those used for server transmission rules. You can construct these queries in the same manner as the condition part of a transmission rule.

Example

The following condition selects messages following the search criteria: priority is set to 4; the originator name is like'%sender%'; and the status is greater than or equal to 20.

```
<condition>
 <priority>4</priority>
 <customRule>ias_Originator LIKE '%sender%' AND ias_Status &gt;= 20</
customRule>
</condition>
```

See also

- "Condition syntax" on page 731

Schedule tag

You can optionally set up server management requests to run on a schedule. Use the following <schedule> subtags to define the schedule on which the request runs.

<schedule> subtags	Description
<starttime>	Defines the time of day at which the server begins generating reports. For example: <starttime>09:00:00</starttime>
<between>	Contains two subtags, starttime and endtime, which define an interval during which the server generates reports. May not be used in the same schedule as starttime. For example: <between> <starttime>Mon Jan 16 09:00:00 EST 2006</starttime> <endtime>Mon Jan 17 09:00:00 EST 2006</endtime> </between>
<everyhour>	Defines the interval between subsequent reports in hours. May not be used in the same schedule as everyminute or everysecond. For example, the following request generates a report every two hours starting at 9 AM: <schedule> <starttime>09:00:00</starttime> <everyhour>2</everyhour> </schedule>
<everyminute>	Defines the interval between subsequent reports in minutes. May not be used in the same schedule as everyhour or everysecond. <schedule> <everyminute>10</everyminute> </schedule>
<everysecond>	Defines the interval between subsequent reports in seconds. May not be used in the same schedule as everyhour or everyminute. <schedule> <everysecond>45</everysecond> </schedule>
<ondayofweek>	Each tag contains one day of the week in which the schedule is active. For example, the following schedule runs on Mondays and Tuesdays: <schedule> <ondayofweek>Monday</ondayofweek> <ondayofweek>Tuesday</ondayofweek> </schedule>
<ondayofmonth>	Each tag contains one day of the month on which the schedule is active. For example, the following schedule runs on the 15th of the month: <schedule> <ondayofmonth>15</ondayofmonth> </schedule>

<schedule> subtags	Description
<startdate>	The date on which the schedule becomes active. For example: <startdate>Mon Jan 16 2006</startdate>

To modify a schedule, register a new server management request with the same requestId. To delete a schedule, register a server management request with the same requestId, but include the schedule tag <schedule>none</schedule>.

Notes

- Each tag, except for the <ondayofweek> and <ondayofmonth> tags, can only be used once in a schedule.
- The <between> tag and the individual <starttime> tag may not both be used in the same schedule.
- Only one of <everysecond>, <everyminute>, and <everyhour> may be used in the same schedule.

Example

The following example creates a persistent schedule that reports on all the messages on the server including the ID and status of each message. It also overwrites any previous persistent requests assigned to the request ID dailyMessageStatus.

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
  <MessageDetailsRequest>
   <request>
     <replyAddr>myclient\messageStatusQueue</replyAddr>
     <requestId>dailyMessageStatus</requestId>
     <schedule>
       <everyhour>24</everyhour>
     </schedule>
     <persistent/>
     <messageId/>
     <status/>
   </request>
  </MessageDetailsRequest>
</actions>
```

The following is an example of what the report might look like. It is sent to the address myclient \messageStatusQueue. It indicates that there are two messages on the server, one with status 60 (received) and one with status 1 (pending).

```
<?xml version="1.0" encoding="UTF-8"?>
<MessageDetailsReport>
   <requestId>dailyMessageStatus</requestId>
   <UTCDatetime>Mon Jan 16 15:03:04 EST 2007</UTCDatetime>
   <statusDescription>Scheduled report</statusDescription>
   <messageCount>2</messageCount>
   <message>
     <messageId>ID:26080b8927f83f9722357eab0a0628eb</messageId>
     <status>60</status>
   </message>
   <message>
```

```
      <messageId>ID:fe857fa8-a7d7-4266-985b-a1818a85d1a2</messageId>
      <status>1</status>
    </message>
  </MessageDetailsReport>
```

MessageDetailsReport tag

Each Message Details Report is an XML message containing the <MessageDetailsReport> tag, and is composed of a report header followed by optional <message> tags. The header of each report consists of the following tags:

<MessageDetailsReport> subtags	Description
<message>	The body of the report consists of a list of <message> tags whose subtags display the specific details of each message that satisfied the selection criteria. If no messages were selected, or no detail elements were specified in the original request, then no <message> tags are included in the report. Otherwise, each message has its own <message> tag.
<messageCount>	The number of messages that satisfy the selection criteria of the request.
<requestId>	The ID of the request that generated the report.
<statusDescription>	A brief description of the reason why this report was generated.
<UTCDateline>	The time and date that this report was generated.

Message tag

<message> subtags	Description
<address>	The address of the message. For example, myclient\myqueue.
<contentSize>	The size of the message content. If the message is a text message, this is the number of characters. If the message is binary, this is the number of bytes.
<expires>	The date and time when the message expires if it is not delivered.
<kind>	Indicates whether the message is binary (1) or text (2).
<messageId>	The message ID of the new message.
<originator>	The message store ID of the originator of the message.

<message> subtags	Description
<priority>	The priority of message: an integer between 0 and 9, where 0 indicates lowest priority and 9 indicates highest priority.
<property>	Properties of the message.
<status>	The current status of the message.
<statusTime>	The time at which the message became its current status. This is the local time.
<transmissionStatus>	The synchronization status of the message. This value can be one of: • 0 - The message has not been transmitted to its intended recipient message store. • 1 - The message has been transmitted to its intended recipient message store. • 2 - The recipient and originating message stores are the same so no transmission is necessary. • 3 - The message has been transmitted to its intended recipient, but that transmission has yet to be confirmed. There is a possibility that the message transmission was interrupted, and that QAnywhere may transmit the message again.

Examples

The following is an example of a message details report:

```
<?xml version="1.0" encoding="UTF-8"?>
<MessageDetailsReport>
 <requestId>testReport</requestId>
 <UTCDatetime>Mon Jan 16 15:03:04 EST 2006</UTCDatetime>
 <statusDescription>Scheduled report</statusDescription>
 <messageCount>1</messageCount>
 <message>
  <messageId>ID:26080b8927f83f9722357eab0a0628eb</messageId>
  <status>60</status>
  <property>
   <name>myPropName</name>
   <value>myPropVal</value>
  </property>
 </message>
</MessageDetailsReport>
```

The following condition selects messages following the search criteria: (msgId=ID:144... OR msgId=ID225...) AND (status=pending) AND (kind=textmessage) AND (contains the property 'myProp' with value 'myVal')

```
<condition>
 <messageId>ID:144d7e44dc2d7e1d</messageId>
 <messageId>ID:22578sd5dsd99s8e</messageId>
 <status>1</status>
 <kind>text</kind>
```

```
 <property>myProp=myVal</property>
</condition>
```

A one-time request is a request that has omitted the <schedule> tag. These requests are used to generate a single report and are deleted when the report has been sent. This request generates a single report that displays the message id, status, and target address of all messages with priority 9 currently on the server.

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
 <MessageDetailsRequest>
  <request>
   <requestId>testRequest</client>
   <condition>
    <priority>9</priority>
   </condition>
   <messageId/>
   <status/>
   <address/>
  </request>
 </MessageDetailsRequest>
</actions>
```

The following sample message details request generates a report that includes the message ID and message status.

```
<?xml version="1.0" encoding="UTF-8"?>
<actions>
  <MessageDetailsRequest>
    <!-- ... -->
    <messageId />
    <status />
  </MessageDetailsRequest>
</actions>
```

See also

- "Message headers" on page 656
- "Message properties" on page 658
- "Predefined message properties" on page 659

Server management request DTD

The following is the complete definition of the server management request XML document type. This DTD is provided as a summary of the server management tags that are described in this section.

```
<!DOCTYPE actions [
<!ELEMENT actions (ActionsResponseId?,(CloseConnector|OpenConnector|
RestartRules|SetProperty
    |ClientStatusRequest|MessageDetailsRequest|CancelMessageRequest|
PauseConnector|ResumeConnector
    |CancelMessages|GetClientList)+)>
<!ELEMENT ActionsResponseId (#PCDATA)>
<!ELEMENT GetClientList EMPTY>
<!ELEMENT CloseConnector (client+)>
<!ELEMENT PauseConnector (client+)>
<!ELEMENT ResumeConnector (client+)>
<!ELEMENT OpenConnector (client+)>
```



```
<!ELEMENT RestartRules (client+)>
<!ELEMENT CancelMessages (client+)>
<!ELEMENT SetProperty (prop+)>
<!ELEMENT prop (client,name,value?)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT value (#PCDATA)>
<!ELEMENT GetProperties (client+)>
<!ELEMENT ClientStatusRequest (requestId?,replyAddr?,client+,(schedule
+,persistent?)?,onEvent*)>
<!ELEMENT onEvent (#PCDATA)>
<!ELEMENT requestId (#PCDATA)>
<!ELEMENT client (#PCDATA)>
<!ELEMENT replyAddr (#PCDATA)>
<!ELEMENT persistent EMPTY>
<!ELEMENT MessageDetailsRequest (requestId?,replyAddr?,(schedule
+,persistent?)?,condition?,

getAddress?,getArchived?,getContentSize?,getExpires?,getKind?,getMessageId?,g
etOriginator?,

getPriority?,getProperties?,getStatus?,getStatusTime?,getTransmissionStatus?
)>
<!ELEMENT getAddress EMPTY>
<!ELEMENT getArchived EMPTY>
<!ELEMENT getContentSize EMPTY>
<!ELEMENT getExpires EMPTY>
<!ELEMENT getKind EMPTY>
<!ELEMENT getMessageId EMPTY>
<!ELEMENT getOriginator EMPTY>
<!ELEMENT getPriority EMPTY>
<!ELEMENT getProperties EMPTY>
<!ELEMENT getStatus EMPTY>
<!ELEMENT getStatusTime EMPTY>
<!ELEMENT getTransmissionStatus EMPTY>
<!ELEMENT CancelMessageRequest (requestId?,replyAddr?,report?,(schedule
+,persistent?)?,condition?,

getAddress?,getArchived?,getContentSize?,getExpires?,getKind?,getMessageId?,g
etOriginator?,

getPriority?,getProperties?,getStatus?,getStatusTime?,getTransmissionStatus?
)>
<!ELEMENT report EMPTY>
<!ELEMENT schedule (((startTime|
between)?,everyHour?,everyMinute?,everySecond?,onDayOfWeek*,
    onDayOfMonth*)?)>
<!ELEMENT between (startTime,endTime)>
<!ELEMENT startTime (#PCDATA)>
<!ELEMENT endTime (#PCDATA)>
<!ELEMENT everyHour (#PCDATA)>
<!ELEMENT everyMinute (#PCDATA)>
<!ELEMENT everySecond (#PCDATA)>
<!ELEMENT onDayOfWeek (#PCDATA)>
<!ELEMENT onDayOfMonth (#PCDATA)>
<!ELEMENT condition ((messageId|status|priority|address|originator|kind|
archived|
    customRule|property)+)>
<!ELEMENT archived EMPTY>
<!ELEMENT messageId (#PCDATA)>
<!ELEMENT status (#PCDATA)>
<!ELEMENT priority (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT originator (#PCDATA)>
<!ELEMENT kind (#PCDATA)>
```

```
<!ELEMENT customRule (#PCDATA)>
<!ELEMENT property (#PCDATA)>
]>
```

QAnywhere Agent utilities reference

qaagent utility

Sends and receives messages for all QAnywhere applications on a single client device. This utility should only be used when the client message store is a SQL Anywhere database.

External utilities, such as dbmlsync, that are used to synchronize message stores are not supported on QAnywhere.

Syntax

qaagent [*option* ...]

Option	Description
@data	Reads options from the specified environment variable or configuration file. See "@data qaagent option" on page 674.
-c *connection-string*	Specifies a connection string to the client message store. See "-c qaagent option" on page 675.
-cd *seconds*	Specifies the delay time between retry attempts to the database. See "-cd qaagent option" on page 676.
-cr *number-of-retries*	Specifies the number of retries to connect to the database after a connection failure. See "-cr qaagent option" on page 676.
-fd *seconds*	Specifies the delay time between retry attempts to the primary MobiLink server. See "-fd qaagent option" on page 677.
-fr *number-of-retries*	Specifies the number of retries to connect to the primary MobiLink server after a connection failure. See "-fr qaagent option" on page 677.
-id *id*	Specifies the ID of the client message store that the QAnywhere Agent is to connect to. See "-id qaagent option" on page 678.
-idl *download-size*	Specifies the maximum size of a download to use during a message transmission. See "-idl qaagent option" on page 679.
-iu *upload-size*	Specifies the maximum size of an upload to use during a message transmission. See "-iu qaagent option" on page 680.

Option	Description
-lp *number*	Specifies the port on which the MobiLink Listener listens for notifications from the MobiLink server. The default is 5001. See "-lp qaagent option" on page 680.
-mn *password*	Specifies a new password for the MobiLink user. See "-mn qaagent option" on page 681.
-mp *password*	Specifies the password for the MobiLink user. See "-mp qaagent option" on page 681.
-mu *username*	Specifies the MobiLink user. See "-mu qaagent option" on page 682.
-o *logfile*	Specifies a file to which to log output messages. See "-o qaagent option" on page 682.
-on *size*	Specifies a maximum size for the QAnywhere Agent message log file, after which the file is renamed with the extension .old and a new file is started. See "-on qaagent option" on page 683.
-os *size*	Specifies a maximum size for the QAnywhere Agent message log file, after which a new log file with a new name is created and used. See "-os qaagent option" on page 683.
-ot *logfile*	Specifies a file to which to log output messages. See "-ot qaagent option" on page 684.
-pc{+|-}	Enables persistent connections for message transmission. See "-pc qaagent option" on page 685.
-policy *policy-type*	Specifies the transmission policy used by the QAnywhere Agent. See "-policy qaagent option" on page 685.
-push *mode*	Enables or disables push notifications. The default is enabled. See "-push qaagent option" on page 687.
-q	Starts the QAnywhere Agent in quiet mode with the window minimized in the system tray. See "-q qaagent option" on page 689.
-qi	Starts the QAnywhere Agent in quiet mode with the window completely hidden. See "-qi qaagent option" on page 689.
-si	Initializes the database for use as a client message store. See "-si qaagent option" on page 690.
-su	Upgrades a client message store to the current version without running dbunload/reload. See "-su qaagent option" on page 691.

Option	Description
-sur	Upgrades a client message store to the current version and performs dbunload/reload of the message store. See "-sur qaagent option" on page 692.
-sv	Uses the SQL Anywhere Network Database Server as the database server. See "-sv qaagent option" on page 693.
-v [*levels*]	Specifies a level of verbosity. See "-v qaagent option" on page 693.
-wc *name*	Specifies the window class name that the QAnywhere Agent or UltraLite Agent is to start. See also "qastop utility" on page 717.
-x { **http**|**tcpip**|**tls**|**https** } [**(***keyword*=*value*;...**)**]	Specifies protocol options for communication with the MobiLink server. See "-x qaagent option" on page 694.
-xd	Specifies that the QAnywhere Agent should use dynamic addressing of the MobiLink server. See "-xd qaagent option" on page 695.

See also

- "Starting the QAnywhere agent" on page 44

@data qaagent option

Reads options from the specified environment variable or configuration file.

Syntax

qaagent @{ *filename* | *environment-variable* } ...

Remarks

With this option, you can put command line options in an environment variable or configuration file. If both exist with the name you specify, the environment variable is used.

If you want to protect passwords or other information in the configuration file, you can use the File Hiding utility to obfuscate the contents of the configuration file.

This option is useful for Windows Mobile because command lines in shortcuts are limited to 256 characters.

Sybase Central equivalent

The QAnywhere plug-in to Sybase Central has tasks called **New Agent Configuration File** (one for SQL Anywhere and one for UltraLite). When you choose one of these, you are prompted to enter a file name and then a **Properties** window appears so that you can enter the command information. The file that is produced has a *.qaa* extension. The *.qaa* file extension is a Sybase Central convention; this file is the

same as what you would create for the @data option. You can use the command file created by Sybase Central as your @data configuration file.

See also

- "Configuration files" [*SQL Anywhere Server - Database Administration*]
- "File Hiding utility (dbfhide)" [*SQL Anywhere Server - Database Administration*]

-c qaagent option

Specifies a string to connect to the client message store.

Syntax

qaagent -c *connection-string* …

Defaults

Connection parameter	Default value
uid	ml_qa_user
pwd	qanywhere

Remarks

The connection string must specify connection parameters in the form *keyword=value*, separated by semicolons, with no spaces between parameters.

ODBC data sources are not typically used on client devices. ODBC is not used by qaagent.

The following are some of the connection parameters you may need to use:

- **dbf=*filename*** Connect to a message store with the specified file name. See "DatabaseFile (DBF) connection parameter" [*SQL Anywhere Server - Database Administration*].
- **dbn=*database-name*** Connect to a client message store that is already running by specifying a database name rather than a database file. See "DatabaseName (DBN) connection parameter" [*SQL Anywhere Server - Database Administration*].
- **server=*server-name*** Specify the name of the database server that is already running. The default value is the name of the database. See "ServerName (Server) connection parameter" [*SQL Anywhere Server - Database Administration*].
- **uid=*user*** Specify a database user ID to connect to the client message store. This parameter is required if you change the default UID or PWD connection parameters. See "Userid (UID) connection parameter" [*SQL Anywhere Server - Database Administration*].
- **pwd=*password*** Specify the password for the database user ID. This is required if you change the default UID or PWD connection parameters. See "Password (PWD) connection parameter" [*SQL Anywhere Server - Database Administration*].

- **dbkey=*key*** Specify the encryption key required to access the database. See "DatabaseKey (DBKEY) connection parameter" [*SQL Anywhere Server - Database Administration*].

- **start=*startline*** Specify the database server start line. If you do not specify the startline, the default for Windows Mobile is `start=dbsrv12 -m -gn 5`, and the default for other Windows platforms is `start=dbsrv12 -m`. The -m option causes the contents of the transaction log to be deleted at checkpoints and is recommended. See:

  - "StartLine (START) connection parameter" [*SQL Anywhere Server - Database Administration*]
  - "-m dbeng12/dbsrv12 server option" [*SQL Anywhere Server - Database Administration*]
  - "-gn dbsrv12 server option" [*SQL Anywhere Server - Database Administration*]

See also

- "Connection parameters" [*SQL Anywhere Server - Database Administration*]
- "SQL Anywhere database connections" [*SQL Anywhere Server - Database Administration*]

Example

```
qaagent -id Device1 -c "DBF=qanyclient.udb" -x tcpip(host=hostname) -policy
automatic
```

-cd qaagent option

When specified in conjunction with the -cr option, this option specifies the delay between attempts to connect to the database.

Syntax

qaagent -cd *seconds* ...

Remarks

The default is a 10 second delay between retries.

If all retries fail, the QAnywhere Agent displays an error and waits to be shut down.

If a database connection fails during operation, the QAnywhere Agent goes through termination steps, including finalizing connections and terminating threads and external processes, and then re-starts.

You must use this option with the qaagent -cr option. The -cr option specifies how many times to retry the connection to the database, and the -cd option specifies the delay between retry attempts.

-cr qaagent option

Specifies the number of times that the QAnywhere Agent should retry the connection to the database.

Syntax

qaagent -cr *number-of-retries* ...

Remarks

The default number of retries is 3, with a 10 second delay between retries.

-fd qaagent option

When specified in conjunction with the -fr option, this option specifies the delay between attempts to connect to the MobiLink server.

Syntax

qaagent -fd *seconds* ...

Default

- If you specify -fr and do not specify -fd, the delay is 0 (no delay between retry attempts).
- If you do not specify -fr, the default is no retry attempts.

Remarks

You must use this option with the qaagent -fr option. The -fr option specifies how many times to retry the connection to the primary server, and the -fd option specifies the delay between retry attempts.

This option is typically used when you specify failover MobiLink servers with the -x option. By default, when you set up a failover MobiLink server, the QAnywhere Agent tries an alternate server immediately upon a failure to reach the primary server. You can use the -fr option to cause the QAnywhere Agent to try the primary server again before going to the alternate server, and you can use the -fd option to specify the amount of time between retries of the primary server.

It is recommended that you set this option to 10 seconds or less.

You cannot use this option with the qaagent -xd option.

See also

- "-fr qaagent option" on page 677
- "-x qaagent option" on page 694
- "Failover mechanism setup" on page 36

-fr qaagent option

Specifies the number of times that the QAnywhere Agent should retry the connection to the primary MobiLink server.

Syntax

qaagent -fr *number-of-retries* ...

Default

0 - the QAnywhere Agent does not attempt to retry the primary MobiLink server.

Remarks

By default, if the QAnywhere Agent is not able to connect to the MobiLink server, there is no error and messages are not sent. This option specifies that the QAnywhere Agent should retry the connection to the MobiLink server, and specifies the number of times that it should retry before trying an alternate server or issuing an error if you have not specified an alternate server.

This option is typically used when you specify failover MobiLink servers with the -x option. By default when you set up a failover MobiLink server, the QAnywhere Agent tries an alternate server immediately upon a failure to reach the primary server. This option causes the QAnywhere Agent to try the primary server again before going to the alternate server.

In addition, you can use the -fd option to specify the amount of time between retries of the primary server.

You cannot use this option with the qaagent -xd option.

See also

- "-fd qaagent option" on page 677
- "-x qaagent option" on page 694
- "Failover mechanism setup" on page 36

-id qaagent option

Specifies the ID of the client message store that the QAnywhere Agent is to connect to.

Syntax

qaagent -id *id* ...

Default

The default value of the ID is the device name on which the Agent is running. You must use the -id option when the device names are not unique.

Remarks

Each client message store is represented by a unique sequence of characters called the message store ID. If you do not supply an ID when you first connect to the message store, the default is the device name. On subsequent connections, you must always specify the same message store ID with the -id option.

The message store ID corresponds to the MobiLink remote ID. It is required because in all MobiLink applications, each remote database must have a unique ID.

If you are starting a second instance of the qaagent on a device, the -id option must be used to specify a unique message store ID.

You cannot use the following characters in an ID:

- double quotes
- control characters

- double backslashes

The following additional constraints apply:

- The ID has a limit of 120 characters.
- You can use a single backslash only if it is used as an escape character.
- If your client message store database has the quoted_identifier database option set to Off, then your ID can only include alphanumeric characters and underscores, at signs, pounds, and dollar signs.

See also

- "MobiLink users" [*MobiLink - Client Administration*]
- "MobiLink user creation and registration" [*MobiLink - Client Administration*]
- "Setting up the client message store" on page 23

-idl qaagent option

Specifies the incremental download size.

Syntax

qaagent -idl *download-size* [**K** | **M**] ...

Default

-1 - no maximum download size.

Remarks

This option specifies the size in bytes of the download part of a message transmission. Use the suffix K or M to specify units of kilobytes or megabytes, respectively.

When the QAnywhere Agent starts, it assigns the value specified by this option to the ias_MaxDownloadSize message store property. This message store property defines an upper bound on the size of a download. When a transmission is triggered, the server tags messages for delivery to the client until the total size of all messages reaches the limit set with this option. The server continues sending batches of messages until all queued messages have been delivered. Transmission rules are re-executed after each batch of messages is transmitted so that if a high priority messages gets queued during a transmission, it jumps to the front of the queue.

Messages queued for delivery that exceed the download threshold are broken into multiple smaller message parts. Each message part can be downloaded separately, resulting in the gradual download of the message over several synchronizations. The complete message arrives at its destination once all of its message parts have arrived.

The incremental download size is an approximation. The actual download size depends on many factors beyond the size of the message.

See also

- ias_MaxDownloadSize in "Predefined client message store properties" on page 717

-iu qaagent option

Specifies the incremental upload size.

Syntax

qaagent -iu *upload-size* [**K** | **M**] ...

Default

256K

Remarks

This option specifies the size in bytes of the upload part of a message transmission. Use the suffix K or M to specify units of kilobytes or megabytes, respectively.

When the QAnywhere Agent starts, it assigns the value specified by this option to the ias_MaxUploadSize message store property. This message store property defines an upper bound on the size of an upload. When a transmission is triggered, the Agent tags messages for delivery to the server until the total size of all messages reaches the limit set with this option. When the limit is reached, these messages are sent to the server. As long as the messages arrive at the server and an acknowledgement is successfully sent from the server to the client, these messages are considered to be successfully delivered, even if the download phase of the transmission fails. The Agent continues sending batches of messages to the server until all queued messages have been delivered. Transmission rules are re-executed after each batch of messages is transmitted so that if a high priority messages gets queued during a transmission, it jumps to the front of the queue.

Messages that exceed the upload threshold are broken into multiple smaller message parts. Each message part can be uploaded separately, resulting in the gradual upload of the message over several synchronizations. The complete message arrives at its destination once all of its message parts have arrived.

The incremental upload size is an approximation. The actual upload size depends on many factors beyond the size of the message.

See also

- ias_MaxUploadSize in "Predefined client message store properties" on page 717

-lp qaagent option

Specifies the MobiLink Listener port.

Syntax

qaagent -lp *number* ...

Default

5001

Remarks

The port number on which the MobiLink Listener listens for UDP notifications from the MobiLink server. Notifications are used to inform the QAnywhere Agent that a message is waiting. The UDP port is also used by the QAnywhere Agent to send control commands to the MobiLink Listener.

See also

- "Scenario for messaging with push notifications" on page 6
- "-push qaagent option" on page 687

-mn qaagent option

Specifies a new password for the MobiLink user.

Syntax

qaagent -mp *password* ...

Default

None

Remarks

Use to change the password.

See also

- "MobiLink users" [*MobiLink - Client Administration*]
- "-mp qaagent option" on page 681
- "-mu qaagent option" on page 682

-mp qaagent option

Specifies the MobiLink password for the MobiLink user.

Syntax

qaagent -mp *password* ...

Default

None

Remarks

If the MobiLink server requires user authentication, use -mp to supply the MobiLink password.

See also

- "MobiLink users" [*MobiLink - Client Administration*]
- "-mu qaagent option" on page 682

-mu qaagent option

Specifies the MobiLink user name.

Syntax

qaagent -mu *username* ...

Default

The client message store ID

Remarks

The MobiLink user name is used for authentication with the MobiLink server.

If you specify a user name that does not exist, it is created for you.

All MobiLink user names must be registered in the server message store. See "Registering QAnywhere client user names" on page 30.

See also

- "MobiLink users" [*MobiLink - Client Administration*]
- "-id qaagent option" on page 678
- "-mp qaagent option" on page 681
- "Remote IDs" [*MobiLink - Client Administration*]

-o qaagent option

Sends output to the specified log file.

Syntax

qaagent -o *logfile* ...

Default

None

Remarks

The QAnywhere Agent logs output to the file name that you specify. If the file already exists, new log information is appended to the file. The SQL Anywhere synchronization client (dbmlsync) logs output to a file with the same name, but including the suffix *_sync*. The MobiLink Listener utility (dblsn) logs output to a file with the same name, but including the suffix *_lsn*.

For example, if you specify the log file *c:\tmp\mylog.out*, then qaagent logs to *c:\tmp\mylog.out*, dbmlsync logs to *c:\tmp\mylog_sync.out*, and dblsn logs to *c:\tmp\mylog_lsn.out*.

See also

- "-ot qaagent option" on page 684
- "-on qaagent option" on page 683
- "-os qaagent option" on page 683
- "-v qaagent option" on page 693

-on qaagent option

Specifies a maximum size for the QAnywhere Agent message log file, after which the file is renamed with the extension *.old* and a new file is started.

Syntax

qaagent -on *size* [**k** | **m**] ...

Default

None

Remarks

The *size* is the maximum file size for the message log, in bytes. Use the suffix k or m to specify units of kilobytes or megabytes, respectively. The minimum size limit is 10k.

When the log file reaches the specified size, the QAnywhere Agent renames the output file with the extension *.old*, and starts a new one with the original name.

Note
If the *.old* file already exists, it is overwritten. To avoid losing old log files, use the -os option instead.

This option cannot be used with the -os option.

See also

- "-o qaagent option" on page 682
- "-ot qaagent option" on page 684
- "-os qaagent option" on page 683
- "-v qaagent option" on page 693

-os qaagent option

Specifies a maximum size for the QAnywhere Agent message log file, after which a new log file with a new name is created and used.

Syntax

qaagent -os *size* [**k** | **m**] ...

Default

None

Remarks

The *size* is the maximum file size for logging output messages. The default units is bytes. Use the suffix k or m to specify units of kilobytes or megabytes, respectively. The minimum size limit is 10k.

Before the QAnywhere Agent logs output messages to a file, it checks the current file size. If the log message makes the file size exceed the specified size, the QAnywhere Agent renames the message log file to *yymmddxx.mls*. In this instance, *xx* are sequential characters ranging from 00 to 99, and *yymmdd* represents the current year, month, and day.

You can use this option to prune old message log files to free up disk space. The latest output is always appended to the file specified by -o or -ot.

Note
This option cannot be used with the -on option.

See also

- "-o qaagent option" on page 682
- "-ot qaagent option" on page 684
- "-on qaagent option" on page 683
- "-v qaagent option" on page 693

-ot qaagent option

Truncates the log file and appends output messages to it.

Syntax

qaagent -ot *logfile* ...

Default

None

Remarks

The QAnywhere Agent logs output to the file name that you specify. If the file exists, it is first truncated to a size of 0. The SQL Anywhere synchronization client (dbmlsync) logs output to a file with the same name, but including the suffix *_sync*. The MobiLink Listener utility (dblsn) logs output to a file with the same name, but including the suffix *_lsn*.

For example, if you specify the log file *c:\tmp\mylog.out*, then qaagent logs to *c:\tmp\mylog.out*, dbmlsync logs to *c:\tmp\mylog_sync.out*, and dblsn logs to *c:\tmp\mylog_lsn.out*.

See also

- "-o qaagent option" on page 682
- "-on qaagent option" on page 683
- "-os qaagent option" on page 683
- "-v qaagent option" on page 693

-pc qaagent option

Maintains a persistent connection to the MobiLink server between synchronizations.

Syntax

qaagent -pc { **+** | **-** } ...

Default

-pc-

Remarks

Enabling persistent connections (-pc+) is useful when network coverage is good and there is heavy message traffic over QAnywhere. In this scenario, you can reduce the network overhead of setting up and taking down a TCP/IP connection every time a message transmission occurs.

Disabling persistent connections (-pc-) is useful in the following scenarios when the client device has a public IP address and is reachable by UDP or SMS:

- The client device is using dial-up networking and connection time charges are an issue.
- There is light message traffic over QAnywhere. Persistent TCP/IP connections consume network server resources, and so could have an impact on scalability.
- The client device network coverage is unreliable. You can use the automatic policy to transmit messages when connection is possible. Trying to maintain persistent connections in this environment is not useful and can waste CPU resources.

See also

- "-push qaagent option" on page 687
- "-pc+ dbmlsync option" [*MobiLink - Client Administration*]

-policy qaagent option

Specifies a policy that determines when message transmission occurs.

Syntax

qaagent -policy *policy-type* ...

policy-type: **ondemand** | **scheduled**[*interval-in-seconds*] | **automatic** | *rules-file*

Defaults

- The default policy type is automatic.
- The default interval for scheduled policies is 900 seconds (15 minutes).

Remarks

QAnywhere uses a policy to determine when message transmission occurs. The *policy-type* can be one of the following values:

- **ondemand** Only transmit messages when the QAnywhere client application makes the appropriate method call.

  The QAManager PutMessage() method causes messages to be queued locally. These messages are not transmitted to the server until the QAManager TriggerSendReceive() method is called. Similarly, messages waiting on the server are not sent to the client until TriggerSendReceive() is called by the client.

  When using the ondemand policy, the application is responsible for causing a message transmission to occur when it receives a push notification from the server. A push notification causes a system message to be delivered to the QAnywhere client. In your application, you may choose to respond to this system message by calling TriggerSendReceive().

  For an example, see "System queue" on page 58.

- **scheduled** When a schedule is specified, every *n* seconds the Agent performs message transmission if any of the following conditions are met:

  - New messages were placed in the client message store since the previous time interval elapsed.
  - A message status change occurred since the previous time interval elapsed. This typically occurs when a message is acknowledged by the application. For more information about acknowledgement, see:
    - "AcknowledgementMode enumeration [QAnywhere .NET]" on page 302
    - "AcknowledgementMode class [QAnywhere C++]" on page 354
    - "AcknowledgementMode interface [QAnywhere Java]" on page 468
  - A push notification was received since the previous time interval elapsed.
  - A network status change notification was received since the previous time interval elapsed.
  - Push notifications are disabled.

  You can call the trigger send/receive method to override the time interval. It forces message transmission to occur before the time interval elapses. See:

  - "QAManagerBase.TriggerSendReceive method [QAnywhere .NET]" on page 265
  - "QAManagerBase.triggerSendReceive method [QAnywhere C++]" on page 424
  - "QAManagerBase.triggerSendReceive method [QAnywhere Java]" on page 545
  - SQL: "ml_qa_triggersendreceive" on page 655

- **automatic** Transmit messages when one of the events described below occurs.

  The QAnywhere Agent attempts to keep message queues as current as possible. Any of the following events cause messages queued on the client to be delivered to the server and messages queued on the server to be delivered to the client:

  - Invoking PutMessage().
  - Invoking TriggerSendReceive().
  - A push notification.

    For information about notifications, see "Scenario for messaging with push notifications" on page 6.
  - A message status change on the client. For example, a status change occurs when an application retrieves a message from a local queue which causes the message status to change from pending to received.

- ***rules-file*** Specifies a client transmission rules file. The transmission rules file can indicate a more complicated set of rules to determine when messages are transmitted.

  See "Client transmission rules" on page 738.

See also

- "Determining when message transmission should occur on the client" on page 46
- "Scenario for messaging with push notifications" on page 6

-push qaagent option

Specifies whether push notifications are enabled.

Syntax

qaagent -push *mode* ...

mode : **none** | **connected** | **disconnected** | **lwpoll**

Default

connected

Options

Mode	Description
none	Push notifications are disabled for this agent. The MobiLink Listener (dblsn) is not started.

Mode	Description
connected	Push notifications are enabled for this agent over TCP/IP with persistent connection. The MobiLink Listener (dblsn) is started by qaagent and attempts to maintain a persistent connection to the MobiLink server. This mode is useful when the client device does not have a public IP address or when the MobiLink server is behind a firewall that does not allow UDP messages out. This is the default.
disconnected	Push notifications are enabled for this agent over UDP without a persistent connection. The MobiLink Listener (dblsn) is started by qaagent but does not maintain a persistent connection to the MobiLink server. Instead, a UDP listener receives push notifications from MobiLink. This mode is useful in the following scenarios when the client device has a public IP address and is reachable by UDP or SMS: • The client device is using dial-up networking and connection time charges are an issue. • There is light message traffic over QAnywhere. Persistent TCP/IP connections consume network server resources, and so could have an impact on scalability. • The client device network coverage is unreliable. You can use the automatic policy to transmit messages when connection is possible. Trying to maintain persistent connections in this environment is not useful and can waste CPU resources.
lwpoll	Push notifications are enabled for this agent using light weight polling. The poll period for the light weight poll can be adjusted by including the desired interval in seconds in square brackets following **lwpoll**. For example, **`-push lwpoll`**`[`*`5`*`]` sets a QAnywhere agent to use light weight polling with an interval of 5 seconds between polls. If no interval is given, the default period is 60 seconds. Clients using light weight polling in a secure environment do not need to register a MobiLink user for the MobiLink Listener, as required by other push notification modes.

Remarks

If you do not want to use notifications, set this option to none. You then do not have to deploy the *dblsn.exe* executable with your clients.

For information about QAnywhere without notifications, see "Simple messaging scenario" on page 5.

If you are using UDP, you cannot use push notifications in disconnected mode with ActiveSync due to the limitations of the UDP implementation of ActiveSync.

See also

- "Push notifications" on page 32
- "-pc qaagent option" on page 685
- "Starting the QAnywhere agent" on page 44
- "Notifications of push notification" on page 60
- "-lp qaagent option" on page 680

-q qaagent option

Starts the QAnywhere Agent in quiet mode with the window minimized in the system tray.

Syntax

qaagent -q ...

Default

None

Remarks

When you start the QAnywhere Agent in quiet mode with -q, the main window is minimized to the system tray. In addition, the database server for the message store is started with the -qi option.

See also

- "-qi qaagent option" on page 689

-qi qaagent option

Starts the QAnywhere Agent in quiet mode with the window completely hidden.

Syntax

qaagent -qi ...

Default

None

Remarks

When you start the QAnywhere Agent in quiet mode, on Windows desktop the main window is minimized to the system tray, and on Windows Mobile the main window is hidden. In addition, the database server for the message store is started with the -qi option.

Quiet mode is useful for some Windows Mobile applications because it prevents an application from being closed when Windows Mobile reaches its limit of 32 concurrent processes. Quiet mode allows the QAnywhere Agent to run like a service.

When in -qi quiet mode, you can only stop the QAnywhere Agent by typing **qastop**.

See also

- "qastop utility" on page 717
- "-q qaagent option" on page 689

-si qaagent option

Initializes the database for use as a client message store.

Syntax

qaagent -c "*connection-string*" -si …

Default

None. You only use this option once, to initialize the client message store.

Remarks

Before using this option, you must create a SQL Anywhere database. When you use -si, the QAnywhere Agent initializes the database with database objects such as QAnywhere system tables; it then exits immediately.

When you run -si, you must specify a connection string with the -c option that indicates which database to initialize. The connection string specified in the -c option should also specify a user ID with DBA privileges. If you do not specify a user ID and password, the default user DBA with password SQL is used.

The -si option creates a database user named ml_qa_user and password qanywhere for the client message store. The user called ml_qa_user has permissions suitable for QAnywhere applications only. If you do not change this database user name and password, then you do not need to specify the pwd or uid in the -c option when you start qaagent. If you change either of them, then you must supply the uid and/or pwd in the -c option on the qaagent command line.

Note
You should change the default passwords. To change them, use the GRANT statement.

The -si option does not provide an ID for the client message store. You can assign an ID using the -id option when you run -si or the next time you run qaagent; or, if you do not do that, qaagent, by default, assigns the device name as the ID.

When a message store is created but is not set up with an ID, QAnywhere applications local to the message store can send and receive messages, but cannot exchange messages with remote QAnywhere applications. Once an ID is assigned, remote messaging may also occur.

The -si and -sil options of qaagent are mutually exclusive. You cannot run qaagent with -si on a database that has been initialized with -sil.

See also

- "Setting up the client message store" on page 23
- "Creating a secure client message store" on page 115
- "Changing a password" [*SQL Anywhere Server - Database Administration*]

Examples

The following command connects to a database called *qaclient.db* and initializes it as a QAnywhere client message store. The QAnywhere Agent immediately exits when the initialization is complete.

```
qaagent -si -c "DBF=qaclient.db"
```

-sil qaagent option

Initializes a SQL Anywhere database for local application-to-application messaging.

Syntax

qaagent -c "*connection-string*" -sil …

Default

None. You only use this option once, to initialize the client message store.

Remarks

This option allows you to use a synchronization database for local-only messaging.

> **Note**
> It is not possible to use a synchronization database for QAnywhere remote messaging. For remote QAnywhere messaging applications, the message store database must be dedicated to QAnywhere.

The -si and -sil options of qaagent are mutually exclusive. You cannot run qaagent with -sil on a database that has been initialized with -si.

See also

- "-si qaagent option" on page 690

Example

Following is an example of a command to initialize a database for local-only messaging:

```
qaagent -c "dbf=qanywhere.db" -sil
```

-su qaagent option

Upgrades a client message store to the current version.

Syntax

qaagent -su -c "*connection-string*" ...

Remarks

This option is useful if you want to perform custom actions after the unload/reload and before the qaagent upgrade. Use the -sur option if you are upgrading from a pre-10.0.0 message store and you want the Agent to automatically perform the unload/reload step for you.

If you are upgrading from a pre-10.0.0 message store, you must first manually unload and reload the message store.

This operation exits when the upgrade is complete.

This operation cannot be undone.

See also

- "-sur qaagent option" on page 692

Example

To upgrade from a version 9 database, first, unload and reload the database:

```
dbunload  -q  -c "UID=dba;PWD=sql;DBF=qanywhere.db" -ar
```

Next, run qaagent with the -su option:

```
qaagent -q -su -c "UID=dba;PWD=sql;DBF=qanywhere.db"
```

-sur qaagent option

Upgrades a client message store to the current version.

Syntax

qaagent -sur -c "*connection-string*" ...

Remarks

Specify the database to upgrade in the connection string. The -sur option automatically unloads the message store, reloads it, and upgrades it.

Automatic unload/reload is not supported on Windows Mobile. For Windows Mobile, you must first manually unload and reload the message store.

The unload/reload is necessary to upgrade message stores between major versions of the product. The unload/reload can be done manually along with the -su option. For example, if you need to perform custom actions after the reload and before the upgrade, use the -su option.

This operation exits when the upgrade is complete.

This operation cannot be undone.

See also

- "-su qaagent option" on page 691

Example

The following example unloads and reloads a version 9.0.2 SQL Anywhere database called qanywhere.db, making it useful with QAnywhere version 12.

```
qaagent -q -sur -c "UID=dba;PWD=sql;DBF=qanywhere.db"
```

-sv qaagent option

Informs the agent to use the SQL Anywhere network database server as the database server instead of using the personal database server.

Syntax

qaagent -sv -c "*connection-string*" ...

Remarks

By default, the qaagent connects to the personal database server using the dbeng12 application. When -sv is specified, the qaagent connects to the network database server using the dbsrv12 application.

See also

- "SQL Anywhere database server syntax" [*SQL Anywhere Server - Database Administration*]

-v qaagent option

Allows you to specify what information is logged to the QAnywhere Agent message log file and displayed in the QAnywhere Agent messages window.

Syntax

qaagent -v *levels* ...

Default

Minimal verbosity

Remarks

The -v option affects the message log files and messages window. You only have a message log file if you specify -o or -ot on the qaagent command line.

A high level of verbosity may affect performance and should normally be used in the development phase only.

If you specify -v alone, a minimal amount of information is logged.

The values of *levels* are as follows. You can use one or more of these options at once; for example, -vlm.

- **+** Turn on all logging options.
- **l** Show all MobiLink Listener logging. This causes the MobiLink Listener (dblsn) to start with verbosity level -v3. See the -v option in the "MobiLink Listener utility for Windows devices (dblsn)" [*MobiLink - Server-Initiated Synchronization*].
- **m** Show all dbmlsync logging. This causes the SQL Anywhere synchronization client (dbmlsync) to start with verbosity level -v+. See the dbmlsync "-v dbmlsync option" [*MobiLink - Client Administration*].
- **n** Show all network status change notifications. the QAnywhere Agent receives these notifications from the MobiLink Listener utility.
- **p** Show all message push notifications. The QAnywhere Agent receives these notifications from the MobiLink Listener utility via the MobiLink server, which includes a MobiLink Notifier.
- **q** Show the SQL that is used to represent the transmission rules.
- **s** Show all the message synchronizations that are initialized by QAnywhere Agent.

See also

- "-o qaagent option" on page 682
- "-ot qaagent option" on page 684
- "-on qaagent option" on page 683
- "-os qaagent option" on page 683

-x qaagent option

Specify the network protocol and the protocol options for communication with the MobiLink server.

Syntax

qaagent -x *protocol* [**(** *protocol-options*;... **)** ...]

protocol: **http**, **tcpip**, **https**, **tls**

protocol-options: *keyword*=*value*

Remarks

For a complete list of *protocol-options*, see "MobiLink client network protocol options" [*MobiLink - Client Administration*].

The -x option is required when the MobiLink server is not on the same device as the QAnywhere Agent.

You can specify -x multiple times. This allows you to set up failover to multiple MobiLink servers. When you set up failover, the QAnywhere Agent attempts to connect to the MobiLink servers in the order in which you enter them on the command line.

The QAnywhere Agent also has a Listener that receives notifications from the MobiLink server that messages are available at the server for transmission to the client. This Listener only uses the first MobiLink server that is specified, and does not fail over to others.

See also

- "MobiLink client network protocol options" [*MobiLink - Client Administration*]
- "Communication stream encryption" on page 116
- "Transport-layer security" [*SQL Anywhere Server - Database Administration*]
- "Failover mechanism setup" on page 36
- "-fd qaagent option" on page 677
- "-fr qaagent option" on page 677

-xd qaagent option

Specify that the QAnywhere Agent should use dynamic addressing of the MobiLink server.

Syntax

qaagent -xd

Remarks

When you specify -xd, the QAnywhere Agent can determine the protocol and address of the MobiLink server based on message store properties. This means that it can dynamically determine the address of a single MobiLink server, where the server address is dependent on the current network that is active for the device where the QAnywhere Agent is running.

The QAnywhere application must initialize message store properties that describe the communication protocol and address of the MobiLink server, and establish a relationship with the currently active network interface. As the mobile device switches between different networks, the QAnywhere Agent detects which network is active and automatically adjusts the communication protocol and address of the MobiLink server—without having to be restarted.

See also

- "Client message store properties" on page 26

Example

The following example sets properties so that the appropriate MobiLink address is used based on the type of network the device is on. For example, if the device is on a LAN the appropriate LAN address is used.

```
QAManager mgr;
...
mgr.SetStringStoreProperty( "LAN.CommunicationAddress",
"host=1.2.3.4;port=10997" );
mgr.SetStringStoreProperty( "LAN.CommunicationType", "tcpip" );
mgr.SetStringStoreProperty( "WAN.CommunicationAddress",
"host=5.6.7.8;port=7777" );
mgr.SetStringStoreProperty( "WAN.CommunicationType", "tcpip" );
mgr.SetStringStoreProperty( "EL3C589 Ethernet Adapter.type", "LAN" );
mgr.SetStringStoreProperty( "Acme Wireless Adapter.type", "WAN" );
```

qauagent utility

Sends and receives messages for all QAnywhere applications on a single client device. This utility should only be used when the client message store is an UltraLite database.

Note
The dbmlsync utility, which is designed to synchronize SQL Anywhere message stores, does not support UltraLite message stores.

Syntax

qauagent [*option* ...]

Option	Description
@data	Reads options from the specified environment variable or configuration file. See "@data qauagent option" on page 698.
-c *connection-string*	Specifies a connection string to the client message store. See "-c qauagent option" on page 699.
-cd *seconds*	Specifies the delay time between retry attempts to the database. See "-cd qaagent option" on page 676.
-cr *number-of-retries*	Specifies the number of retries to connect to the database after a connection failure. See "-cr qaagent option" on page 676.
-fd *seconds*	Specifies the delay time between retry attempts to the primary server. See "-fd qauagent option" on page 700.
-fr *number-of-retries*	Specifies the number of retries to connect to the primary server after a connection failure. See "-fr qauagent option" on page 701.
-id *id*	Specifies the ID of the client message store that the QAnywhere UltraLite Agent is to connect to. See "-id qauagent option" on page 702.
-idl *download-size*	Specifies the maximum size of a download to use during a message transmission. See "-idl qauagent option" on page 703.
-iu *upload-size*	Specifies the maximum size of an upload to use during a message transmission. See "-iu qauagent option" on page 703.
-lp *number*	Specifies the port on which the MobiLink Listener listens for notifications from the MobiLink server. The default is 5001. See "-lp qauagent option" on page 704.

Option	Description
-mn *password*	Specifies a new password for the MobiLink user. See "-mn qauagent option" on page 705.
-mp *password*	Specifies the password for the MobiLink user. See "-mp qauagent option" on page 705.
-mu *username*	Specifies the MobiLink user. See "-mp qauagent option" on page 705.
-o *logfile*	Specifies a file to which to log output messages. See "-o qauagent option" on page 706.
-on *size*	Specifies a maximum size for the QAnywhere UltraLite Agent message log file, after which the file is renamed with the extension .old and a new file is started. See "-on qauagent option" on page 707.
-os *size*	Specifies a maximum size for the QAnywhere UltraLite Agent message log file, after which a new log file with a new name is created and used. See "-os qauagent option" on page 707.
-ot *logfile*	Specifies a file to which to log output messages. See "-ot qauagent option" on page 708.
-policy *policy-type*	Specifies the transmission policy used by the QAnywhere UltraLite Agent. See "-policy qauagent option" on page 708.
-push *mode*	Enables or disables push notifications. The default is enabled. See "-push qauagent option" on page 710.
-q	Starts the QAnywhere UltraLite Agent in quiet mode with the window minimized in the system tray. See "-q qauagent option" on page 711.
-qi	Starts the QAnywhere UltraLite Agent in quiet mode with the window completely hidden. See "-qi qauagent option" on page 712.
-si	Initializes the database for use as a client message store. See "-si qauagent option" on page 712.
-sil	Initializes a SQL Anywhere database for local application-to-application messaging. See "-sil qaagent option" on page 691.
-su	Upgrades a client message store to the current version without running dbunload/reload. See "-su qauagent option" on page 713.
-v [*levels*]	Specifies a level of verbosity. See "-v qauagent option" on page 714.

Option	Description
-wc *name*	Specifies the window class name that the QAnywhere UltraLite Agent is to start. See also "qastop utility" on page 717.
-x { **http**|**tcpip**|**tls**|**https** } [**(***keyword=value*;...**)**]	Specifies protocol options for communication with the MobiLink server. See "-x qauagent option" on page 715.
-xd	Specifies that the QAnywhere UltraLite Agent should use dynamic addressing of the MobiLink server. See "-xd qauagent option" on page 716.

See also

- "Starting the QAnywhere agent" on page 44

@data qauagent option

Reads options from the specified environment variable or configuration file.

Syntax

qauagent @{ *filename* | *environment-variable* } ...

Remarks

With this option, you can put command line options in an environment variable or configuration file. If both exist with the name you specify, the environment variable is used.

If you want to protect passwords or other information in the configuration file, you can use the File Hiding utility to obfuscate the contents of the configuration file.

This option is useful for Windows Mobile because command lines in shortcuts are limited to 256 characters.

Sybase Central equivalent

The QAnywhere 12 plug-in to Sybase Central has a task called **Create An Agent Command File**. When you choose it, you are prompted to enter a file name and then a **Properties** window appears that helps you enter the command information. The file that is produced has a *.qaa* extension. The *.qaa* file extension is a Sybase Central convention; this file is the same as what you would create for the @data option. You can use the command file created by Sybase Central as your @data configuration file.

See also

- "Configuration files" [*SQL Anywhere Server - Database Administration*]
- "File Hiding utility (dbfhide)" [*SQL Anywhere Server - Database Administration*]

-c qauagent option

Specifies a string to connect to the client message store.

Syntax

qauagent -c *connection-string* …

Defaults

Connection parameter	Default value
uid	ml_qa_user
pwd	qanywhere

Remarks

The connection string must specify connection parameters in the form *keyword=value*, separated by semicolons, with no spaces between parameters.

ODBC data sources are not typically used on client devices. ODBC is not used by qauagent.

The following are some of the connection parameters you may need to use:

- **dbf=*filename*** Connect to a message store with the specified file name.
- **dbn=*database-name*** Connect to a client message store that is already running by specifying a database name rather than a database file.
- **uid=*user*** Specify a database user ID to connect to the client message store. This parameter is required if you change the default UID or PWD connection parameters.
- **pwd=*password*** Specify the password for the database user ID. This is required if you change the default UID or PWD connection parameters.
- **dbkey=*key*** Specify the encryption key required to access the database.

See also

- "Connection parameters" [*SQL Anywhere Server - Database Administration*]
- "SQL Anywhere database connections" [*SQL Anywhere Server - Database Administration*]
- "UltraLite DBF connection parameter" [*UltraLite - Database Management and Reference*]
- "UltraLite PWD connection parameter" [*UltraLite - Database Management and Reference*]
- "UltraLite DBKEY connection parameter" [*UltraLite - Database Management and Reference*]

Example

```
qauagent -id Device1 -c "DBF=qanyclient.db" -x tcpip(host=hostname) -policy
automatic
```

-cd qauagent option

When specified in conjunction with the -cr option, this option specifies the delay between attempts to connect to the database.

Syntax

qauagent -cd *seconds* ...

Remarks

The default is a 10 second delay between retries.

If all retries fail, the QAnywhere Agent displays an error and waits to be shut down.

If a database connection fails during operation, the QAnywhere Agent goes through termination steps, including finalizing connections and terminating threads and external processes, and then re-starts.

You must use this option with the qauagent -cr option. The -cr option specifies how many times to retry the connection to the database, and the -cd option specifies the delay between retry attempts.

-cr qauagent option

Specifies the number of times that the QAnywhere UltraLite Agent should retry the connection to the database.

Syntax

qauagent -cr *number-of-retries* ...

Remarks

The default number of retries is 3, with a 10 second delay between retries.

-fd qauagent option

When specified in conjunction with the -fr option, this option specifies the delay between attempts to connect to the MobiLink server.

Syntax

qauagent -fd *seconds* ...

Default

- If you specify -fr and do not specify -fd, the delay is 0 (no delay between retry attempts).
- If you do not specify -fr, the default is no retry attempts.

Remarks

You must use this option with the qauagent -fr option. The -fr option specifies how many times to retry the connection to the primary server, and the -fd option specifies the delay between retry attempts.

This option is typically used when you specify failover MobiLink servers with the -x option. By default, when you set up a failover MobiLink server, the QAnywhere UltraLite Agent tries an alternate server immediately upon a failure to reach the primary server. You can use the -fr option to cause the QAnywhere UltraLite Agent to try the primary server again before going to the alternate server, and you can use the -fd option to specify the amount of time between retries of the primary server.

It is recommended that you set this option to 10 seconds or less.

You cannot use this option with the qauagent -xd option.

See also

- "-fr qauagent option" on page 701
- "-x qauagent option" on page 715
- "Failover mechanism setup" on page 36

-fr qauagent option

Specifies the number of times that the QAnywhere UltraLite Agent should retry the connection to the primary MobiLink server.

Syntax

qauagent -fr *number-of-retries* ...

Default

0 - the QAnywhere UltraLite Agent does not attempt to retry the primary MobiLink server.

Remarks

By default, if the QAnywhere UltraLite Agent is not able to connect to the MobiLink server, there is no error and messages are not sent. This option specifies that the QAnywhere UltraLite Agent should retry the connection to the MobiLink server, and specifies the number of times that it should retry before trying an alternate server or issuing an error if you have not specified an alternate server.

This option is typically used when you specify failover MobiLink servers with the -x option. By default when you set up a failover MobiLink server, the QAnywhere UltraLite Agent tries an alternate server immediately upon a failure to reach the primary server. This option causes the QAnywhere UltraLite Agent to try the primary server again before going to the alternate server.

In addition, you can use the -fd option to specify the amount of time between retries of the primary server.

You cannot use this option with the qauagent -xd option.

See also

- "-fd qauagent option" on page 700
- "-x qauagent option" on page 715
- "Failover mechanism setup" on page 36

-id qauagent option

Specifies the ID of the client message store that the QAnywhere UltraLite Agent is to connect to.

Syntax

qauagent -id *id* ...

Default

The default value of the ID is the device name on which the Agent is running. You must use the -id option when the device names are not unique.

Remarks

Each client message store is represented by a unique sequence of characters called the message store ID. If you do not supply an ID when you first connect to the message store, the default is the device name. On subsequent connections, you must always specify the same message store ID with the -id option.

The message store ID corresponds to the MobiLink remote ID. It is required because in all MobiLink applications, each remote database must have a unique ID.

If you are starting a second instance of the qauagent on a device, the -id option must be used to specify a unique message store ID.

You cannot use the following characters in an ID:

- double quotes
- control characters
- double backslashes

The following additional constraints apply:

- The ID has a limit of 120 characters.
- You can use a single backslash only if it is used as an escape character.
- If your client message store database has the quoted_identifier database option set to Off, then your ID can only include alphanumeric characters and underscores, at signs, pounds, and dollar signs.

See also

- "MobiLink users" [*MobiLink - Client Administration*]
- "MobiLink user creation and registration" [*MobiLink - Client Administration*]
- "Setting up the client message store" on page 23

-idl qauagent option

Specifies the incremental download size.

Syntax

qauagent -idl *download-size* [**K** | **M**] ...

Default

-1 - no maximum download size.

Remarks

This option specifies the size in bytes of the download part of a message transmission. Use the suffix K or M to specify units of kilobytes or megabytes, respectively.

When the QAnywhere UltraLite Agent starts, it assigns the value specified by this option to the ias_MaxDownloadSize message store property. This message store property defines an upper bound on the size of a download. When a transmission is triggered, the server tags messages for delivery to the client until the total size of all messages reaches the limit set with this option. The server continues sending batches of messages until all queued messages have been delivered. Transmission rules are re-executed after each batch of messages is transmitted so that if a high priority messages gets queued during a transmission, it jumps to the front of the queue.

Messages queued for delivery that exceed the download threshold are broken into multiple smaller message parts. Each message part can be downloaded separately, resulting in the gradual download of the message over several synchronizations. The complete message arrives at its destination once all of its message parts have arrived.

The incremental download size is an approximation. The actual download size depends on many factors beyond the size of the message.

See also

- ias_MaxDownloadSize in "Predefined client message store properties" on page 717

-iu qauagent option

Specifies the incremental upload size.

Syntax

qauagent -iu *upload-size* [**K** | **M**] ...

Default

256K

Remarks

This option specifies the size in bytes of the upload part of a message transmission. Use the suffix K or M to specify units of kilobytes or megabytes, respectively.

When the QAnywhere UltraLite Agent starts, it assigns the value specified by this option to the ias_MaxUploadSize message store property. This message store property defines an upper bound on the size of an upload. When a transmission is triggered, the Agent tags messages for delivery to the server until the total size of all messages reaches the limit set with this option. When the limit is reached, these messages are sent to the server. As long as the messages arrive at the server and an acknowledgement is successfully sent from the server to the client, these messages are considered to be successfully delivered, even if the download phase of the transmission fails. The Agent continues sending batches of messages to the server until all queued messages have been delivered. Transmission rules are re-executed after each batch of messages is transmitted so that if a high priority messages gets queued during a transmission, it jumps to the front of the queue.

Messages that exceed the upload threshold are broken into multiple smaller message parts. Each message part can be uploaded separately, resulting in the gradual upload of the message over several synchronizations. The complete message arrives at its destination once all of its message parts have arrived.

The incremental upload size is an approximation. The actual upload size depends on many factors beyond the size of the message.

See also

- ias_MaxUploadSize in "Predefined client message store properties" on page 717

-lp qauagent option

Specifies the MobiLink Listener port.

Syntax

qauagent -lp *number* ...

Default

5001

Remarks

The port number on which the MobiLink Listener listens for UDP notifications from the MobiLink server. Notifications are used to inform the QAnywhere UltraLite Agent that a message is waiting. The UDP port is also used by the QAnywhere UltraLite Agent to send control commands to the MobiLink Listener.

See also

- "Scenario for messaging with push notifications" on page 6
- "-push qauagent option" on page 710

-mn qauagent option

Specifies a new password for the MobiLink user.

Syntax

qauagent -mp *password* ...

Default

None

Remarks

Use to change the password.

See also

- "MobiLink users" [*MobiLink - Client Administration*]
- "-mp qauagent option" on page 705
- "-mu qauagent option" on page 705

-mp qauagent option

Specifies the MobiLink password for the MobiLink user.

Syntax

qauagent -mp *password* ...

Default

None

Remarks

If the MobiLink server requires user authentication, use -mp to supply the MobiLink password.

See also

- "MobiLink users" [*MobiLink - Client Administration*]
- "-mu qauagent option" on page 705

-mu qauagent option

Specifies the MobiLink user name.

Syntax

qauagent -mu *username* ...

Default

The client message store ID

Remarks

The MobiLink user name is used for authentication with the MobiLink server.

If you specify a user name that does not exist, it is created for you.

All MobiLink user names must be registered in the server message store. See "Registering QAnywhere client user names" on page 30.

See also

- "MobiLink users" [*MobiLink - Client Administration*]
- "-id qauagent option" on page 702
- "-mp qauagent option" on page 705
- "Remote IDs" [*MobiLink - Client Administration*]

-o qauagent option

Sends output to the specified log file.

Syntax

qauagent -o *logfile* ...

Default

None

Remarks

The QAnywhere UltraLite Agent logs output to the file name that you specify. If the file already exists, new log information is appended to the file. The MobiLink Listener utility (dblsn) logs output to a file with the same name, but including the suffix *_lsn*.

For example, if you specify the log file *c:\tmp\mylog.out*, then qauagent logs to *c:\tmp\mylog.out*, and dblsn logs to *c:\tmp\mylog_lsn.out*.

See also

- "-ot qauagent option" on page 708
- "-on qauagent option" on page 707
- "-os qauagent option" on page 707
- "-v qauagent option" on page 714

-on qauagent option

Specifies a maximum size for the QAnywhere UltraLite Agent message log file, after which the file is renamed with the extension *.old* and a new file is started.

Syntax

qauagent -on *size* [**k** | **m**] ...

Default

None

Remarks

The *size* is the maximum file size for the message log, in bytes. Use the suffix k or m to specify units of kilobytes or megabytes, respectively. The minimum size limit is 10k.

When the log file reaches the specified size, the QAnywhere UltraLite Agent renames the output file with the extension *.old*, and starts a new one with the original name.

Note
If the *.old* file already exists, it is overwritten. To avoid losing old log files, use the -os option instead.

This option cannot be used with the -os option.

See also

- "-o qauagent option" on page 706
- "-ot qauagent option" on page 708
- "-os qauagent option" on page 707
- "-v qauagent option" on page 714

-os qauagent option

Specifies a maximum size for the QAnywhere UltraLite Agent message log file, after which a new log file with a new name is created and used.

Syntax

qauagent -os *size* [**k** | **m**] ...

Default

None

Remarks

The *size* is the maximum file size for logging output messages. The default units is bytes. Use the suffix k or m to specify units of kilobytes or megabytes, respectively. The minimum size limit is 10k.

Before the QAnywhere UltraLite Agent logs output messages to a file, it checks the current file size. If the log message makes the file size exceed the specified size, the QAnywhere UltraLite Agent renames the message log file to *yymmddxx.mls*. In this instance, *xx* are sequential characters ranging from 00 to 99, and *yymmdd* represents the current year, month, and day.

You can use this option to prune old message log files to free up disk space. The latest output is always appended to the file specified by -o or -ot.

Note
This option cannot be used with the -on option.

See also

- "-o qauagent option" on page 706
- "-ot qauagent option" on page 708
- "-on qauagent option" on page 707
- "-v qauagent option" on page 714

-ot qauagent option

Truncates the log file and appends output messages to it.

Syntax

qauagent -ot *logfile* ...

Default

None

Remarks

The QAnywhere UltraLite Agent logs output to the file name that you specify. If the file exists, it is first truncated to a size of 0. The MobiLink Listener utility (dblsn) logs output to a file with the same name, but including the suffix *_lsn*.

For example, if you specify the log file *c:\tmp\mylog.out*, then qauagent logs to *c:\tmp\mylog.out*, and dblsn logs to *c:\tmp\mylog_lsn.out*.

See also

- "-o qauagent option" on page 706
- "-on qauagent option" on page 707
- "-os qauagent option" on page 707
- "-v qauagent option" on page 714

-policy qauagent option

Specifies a policy that determines when message transmission occurs.

Syntax

qauagent -policy *policy-type* ...

policy-type: **ondemand** | **scheduled**[*interval-in-seconds*] | **automatic** | *rules-file*

Defaults

- The default policy type is automatic.
- The default interval for scheduled policies is 900 seconds (15 minutes).

Remarks

QAnywhere uses a policy to determine when message transmission occurs. The *policy-type* can be one of the following values:

- **ondemand** Only transmit messages when the QAnywhere client application makes the appropriate method call.

  The QAManager PutMessage() method causes messages to be queued locally. These messages are not transmitted to the server until the QAManager TriggerSendReceive() method is called. Similarly, messages waiting on the server are not sent to the client until TriggerSendReceive() is called by the client.

  When using the ondemand policy, the application is responsible for causing a message transmission to occur when it receives a push notification from the server. A push notification causes a system message to be delivered to the QAnywhere client. In your application, you may choose to respond to this system message by calling TriggerSendReceive().

  For an example, see "System queue" on page 58.

- **scheduled** Transmit messages at a specified interval. The default value is 900 seconds (15 minutes).

  When a schedule is specified, every *n* seconds the Agent performs message transmission if any of the following conditions are met:

  - New messages were placed in the client message store since the previous time interval elapsed.
  - A message status change occurred since the previous time interval elapsed. This typically occurs when a message is acknowledged by the application.
  - A push notification was received since the previous time interval elapsed.
  - A network status change notification was received since the previous time interval elapsed.
  - Push notifications are disabled.

  You can call the trigger send/receive method to override the time interval. It forces message transmission to occur before the time interval elapses.

- **automatic** Transmit messages when one of the events described below occurs.

The QAnywhere UltraLite Agent attempts to keep message queues as current as possible. Any of the following events cause messages queued on the client to be delivered to the server and messages queued on the server to be delivered to the client:

- Invoking PutMessage().
- Invoking TriggerSendReceive().
- A push notification.
- A message status change on the client. For example, a status change occurs when an application retrieves a message from a local queue which causes the message status to change from pending to received.

- ***rules-file*** Specifies a client transmission rules file. The transmission rules file can indicate a more complicated set of rules to determine when messages are transmitted.

See also

- "Determining when message transmission should occur on the client" on page 46
- "Scenario for messaging with push notifications" on page 6
- "Client transmission rules" on page 738

-push qauagent option

Specifies whether push notifications are enabled.

Syntax

qauagent -push *mode* ...

mode : **none** | **connected** | **disconnected** | **lwpoll**

Default

connected

Options

Mode	Description
none	Push notifications are disabled for this agent. The MobiLink Listener (dblsn) is not started.
connected	Push notifications are enabled for this agent over TCP/IP with persistent connection. The MobiLink Listener (dblsn) is started by qauagent and attempts to maintain a persistent connection to the MobiLink server. This mode is useful when the client device does not have a public IP address or when the MobiLink server is behind a firewall that does not allow UDP messages out. This is the default.

Mode	Description
disconnected	Push notifications are enabled for this agent over UDP without a persistent connection. The MobiLink Listener (dblsn) is started by qauagent but does not maintain a persistent connection to the MobiLink server. Instead, a UDP listener receives push notifications from MobiLink. This mode is useful in the following scenarios when the client device has a public IP address and is reachable by UDP or SMS: • The client device is using dial-up networking and connection time charges are an issue. • There is light message traffic over QAnywhere. Persistent TCP/IP connections consume network server resources, and so could have an impact on scalability. • The client device network coverage is unreliable. You can use the automatic policy to transmit messages when connection is possible. Trying to maintain persistent connections in this environment is not useful and can waste CPU resources.
lwpoll	Push notifications are enabled for this agent using light weight polling. The poll period for the light weight poll can be adjusted by including the desired interval in seconds in square brackets following **lwpoll**. For example, **`-push lwpoll`**`[5]` sets a QAnywhere agent to use light weight polling with an interval of 5 seconds between polls. If no interval is given, the default period is 60 seconds. Clients using light weight polling in a secure environment do not need to register a MobiLink user for the MobiLink Listener, as required by other push notification modes.

Remarks

If you do not want to use notifications, set this option to none. You then do not have to deploy the *dblsn.exe* executable with your clients.

For information about QAnywhere without notifications, see "Simple messaging scenario" on page 5.

If you are using UDP, you cannot use push notifications in disconnected mode with ActiveSync due to the limitations of the UDP implementation of ActiveSync.

See also

- "Push notifications" on page 32
- "Starting the QAnywhere agent" on page 44
- "Notifications of push notification" on page 60
- "-lp qauagent option" on page 704

-q qauagent option

Starts the QAnywhere UltraLite Agent in quiet mode with the window minimized in the system tray.

Syntax

qauagent -q …

Default

None

Remarks

When you start the QAnywhere UltraLite Agent in quiet mode with -q, the main window is minimized to the system tray. In addition, the database server for the message store is started with the -qi option.

See also

- "-qi qauagent option" on page 712

-qi qauagent option

Starts the QAnywhere UltraLite Agent in quiet mode with the window completely hidden.

Syntax

qauagent -qi …

Default

None

Remarks

When you start the QAnywhere UltraLite Agent in quiet mode, on Windows desktop the main window is minimized to the system tray, and on Windows Mobile the main window is hidden. In addition, the database server for the message store is started with the -qi option.

Quiet mode is useful for some Windows Mobile applications because it prevents an application from being closed when Windows Mobile reaches its limit of 32 concurrent processes. Quiet mode allows the QAnywhere UltraLite Agent to run like a service.

When in -qi quiet mode, you can only stop the QAnywhere UltraLite Agent by typing **qastop**.

See also

- "qastop utility" on page 717
- "-q qauagent option" on page 711

-si qauagent option

Initializes the database for use as a client message store.

Syntax

qauagent -c "*connection-string*" -si …

Default

None. You only use this option once, to initialize the client message store.

Remarks

Before using this option, you must create an UltraLite database. When you use -si, the QAnywhere UltraLite Agent initializes the database with database objects such as QAnywhere system tables; it then exits immediately.

When you run -si, you must specify a connection string with the -c option that indicates which database to initialize. The connection string specified in the -c option should also specify a user ID with DBA privileges. If you do not specify a user ID and password, the default user DBA with password SQL is used.

The -si option creates a database user named ml_qa_user and password qanywhere for the client message store. The user called ml_qa_user has permissions suitable for QAnywhere applications only. If you do not change this database user name and password, then you do not need to specify the pwd or uid in the -c option when you start qauagent. If you change either of them, then you must supply the uid and/or pwd in the -c option on the qauagent command line.

Note
You should change the default passwords. To change them, use the GRANT statement.

The -si option does not provide an ID for the client message store. You can assign an ID using the -id option when you run -si or the next time you run qauagent; or, if you do not do that, qauagent, by default, assigns the device name as the ID.

When a message store is created but is not set up with an ID, QAnywhere applications local to the message store can send and receive messages, but cannot exchange messages with remote QAnywhere applications. Once an ID is assigned, remote messaging may also occur.

See also

- “Setting up the client message store” on page 23
- “Creating a secure client message store” on page 115
- “Changing a password” [*SQL Anywhere Server - Database Administration*]

Examples

The following command connects to a database called *qaclient.db* and initializes it as a QAnywhere client message store. The QAnywhere UltraLite Agent immediately exits when the initialization is complete.

```
qauagent -si -c "DBF=qaclient.udb"
```

-su qauagent option

Upgrades a client message store to the current version. If you are upgrading from a pre-10.0.0 message store, you must first manually unload and reload the message store.

Syntax

qauagent -su -c "*connection-string*" ...

Remarks

This option is useful if you want to perform custom actions after the unload/reload and before the qauagent upgrade. Use the -sur option if you are upgrading from a pre-10.0.0 message store and you want the Agent to automatically perform the unload/reload step for you.

This operation exits when the upgrade is complete.

This operation cannot be undone.

Example

To upgrade from a version 9 database, first, unload and reload the database:

```
dbunload  -q  -c "UID=dba;PWD=sql;DBF=qanywhere.db" -ar
```

Next, run qauagent with the -su option:

```
qauagent -q -su -c "UID=dba;PWD=sql;DBF=qanywhere.db"
```

-v qauagent option

Allows you to specify what information is logged to the message log file and displayed in the QAnywhere UltraLite Agent console.

Syntax

qauagent -v *levels* ...

Default

Minimal verbosity

Remarks

The -v option affects the log files and console. You only have a message log if you specify -o or -ot on the qauagent command line.

A high level of verbosity may affect performance and should normally be used in the development phase only.

If you specify -v alone, a minimal amount of information is logged.

The values of *levels* are as follows. You can use one or more of these options at once; for example, -vlm.

- **+** Turn on all logging options.
- **l** Show all MobiLink Listener logging. This causes the MobiLink Listener (dblsn) to start with verbosity level -v3. See the -v option in the "MobiLink Listener utility for Windows devices (dblsn)" [*MobiLink - Server-Initiated Synchronization*].

- **m** Show all synchronization logging.
- **n** Show all network status change notifications. the QAnywhere UltraLite Agent receives these notifications from the MobiLink Listener utility.
- **p** Show all message push notifications. The QAnywhere UltraLite Agent receives these notifications from the MobiLink Listener utility via the MobiLink server, which includes a MobiLink Notifier.
- **q** Show the SQL that is used to represent the transmission rules.
- **s** Show all the message synchronizations that are initialized by QAnywhere UltraLite Agent.

See also

- "-o qauagent option" on page 706
- "-ot qauagent option" on page 708
- "-on qauagent option" on page 707
- "-os qauagent option" on page 707

-x qauagent option

Specify the network protocol and the protocol options for communication with the MobiLink server.

Syntax

qauagent -x *protocol* [**(** *protocol-options*;... **)** ...]

protocol: **http**, **tcpip**, **https**, **tls**

protocol-options: *keyword*=*value*

Remarks

For a complete list of *protocol-options*, see "MobiLink client network protocol options" [*MobiLink - Client Administration*].

The -x option is required when the MobiLink server is not on the same device as the QAnywhere UltraLite Agent.

You can specify -x multiple times. This allows you to set up failover to multiple MobiLink servers. When you set up failover, the QAnywhere UltraLite Agent attempts to connect to the MobiLink servers in the order in which you enter them on the command line.

The QAnywhere UltraLite Agent also has a Listener that receives notifications from the MobiLink server that messages are available at the server for transmission to the client. This Listener only uses the first MobiLink server that is specified, and does not fail over to others.

See also

- "MobiLink client network protocol options" [*MobiLink - Client Administration*]
- "Communication stream encryption" on page 116
- "Transport-layer security" [*SQL Anywhere Server - Database Administration*]
- "Failover mechanism setup" on page 36
- "-fd qauagent option" on page 700
- "-fr qauagent option" on page 701

-xd qauagent option

Specify that the QAnywhere UltraLite Agent should use dynamic addressing of the MobiLink server.

Syntax

qauagent -xd

Remarks

When you specify -xd, the QAnywhere UltraLite Agent can determine the protocol and address of the MobiLink server based on message store properties. This means that it can dynamically determine the address of a single MobiLink server, where the server address is dependent on the current network that is active for the device where the QAnywhere UltraLite Agent is running.

The QAnywhere application must initialize message store properties that describe the communication protocol and address of the MobiLink server, and establish a relationship with the currently active network interface. As the mobile device switches between different networks, the QAnywhere UltraLite Agent detects which network is active and automatically adjusts the communication protocol and address of the MobiLink server—without having to be restarted.

See also

- "Client message store properties" on page 26

Example

The following example sets properties so that the appropriate MobiLink address is used based on the type of network the device is on. For example, if the device is on a LAN the appropriate LAN address is used.

```
QAManager mgr;
...
mgr.SetStringStoreProperty( "LAN.CommunicationAddress",
"host=1.2.3.4;port=10997" );
mgr.SetStringStoreProperty( "LAN.CommunicationType", "tcpip" );
mgr.SetStringStoreProperty( "WAN.CommunicationAddress",
"host=5.6.7.8;port=7777" );
mgr.SetStringStoreProperty( "WAN.CommunicationType", "tcpip" );
mgr.SetStringStoreProperty( "EL3C589 Ethernet Adapter.type", "LAN" );
mgr.SetStringStoreProperty( "Acme Wireless Adapter.type", "WAN" );
```


qastop utility

Stops the QAnywhere Agent or QAnywhere UltraLite Agent when the agent is running in quiet mode.

Syntax

qastop [*option* ...]

Option	Description
-id *id*	Specifies the ID of the client message store that the QAnywhere Agent or UltraLite Agent is to stop.
-wc *name*	Specifies the window class name that the QAnywhere Agent or UltraLite Agent is to stop.

See also

- "-q qaagent option" on page 689
- "-q qauagent option" on page 711
- "Stopping the QAnywhere Agent" on page 45

QAnywhere properties

Client message store properties

The following sections provide information about client message store properties.

Predefined client message store properties

Several client message store properties have been predefined for your convenience. The predefined message store properties are:

- **ias_Adapters** A list of network adapters that can be used to connect to the MobiLink server. The list is a string and is delimited by a vertical bar.
- **ias_MaxDeliveryAttempts** When defined, the maximum number of times that a message can be received without being acknowledged before its status is set to UNRECEIVABLE. By default, this property is not defined and is equivalent to a value of -1, which means that the client library continues to attempt to deliver an unacknowledged message forever.
- **ias_MaxDownloadSize** The download increment size. By default, QAnywhere uses a maximum download size of -1 which means there is no maximum. If a message originating from a server connector or destination alias exceeds the download increment size specified, the message is broken into smaller message parts and sent in separate downloads. This property is set by the qaagent -idl

option. See "-idl qaagent option" on page 679 for SQL Anywhere message stores and "-idl qauagent option" on page 703 for UltraLite message stores.

- **ias_MaxUploadSize** The upload increment size. By default, QAnywhere uploads messages in increments of 256K. If a message exceeds the upload increment size specified, the message is broken into smaller message parts and sent in separate uploads. This property is set by the qaagent -iu option. See "-iu qaagent option" on page 680 for SQL Anywhere message stores and "-iu qauagent option" on page 703 for UltraLite message stores.

- **ias_Network** Information about the current network in use. This property can be read but should not be set. ias_Network is a special property. It has several built-in attributes that provide information regarding the current network that is being used by the device. The following attributes are automatically set by QAnywhere:

  - **ias_Network.Adapter** The current name of the network card, if any. (The name of the network card that is assigned to the Adapter attribute is displayed in the Agent window when the network connection is established.)

  - **ias_Network.RAS** The current RAS entry name, if any.

  - **ias_Network.IP** The current IP address assigned to the device, if any.

  - **ias_Network.MAC** The current MAC address of the network card being used, if any.

- **ias_RASNames** String. A list of RAS entry names that can be used to connect to the MobiLink server. The list is delimited by a vertical bar.

- **ias_StoreID** The message store ID.

- **ias_StoreInitialized** True if this message stores has successfully been initialized for QAnywhere messaging; otherwise False.

  See "-si qaagent option" on page 690 for SQL Anywhere message stores and "-si qauagent option" on page 712 for UltraLite message stores.

- **ias_StoreVersion** The QAnywhere-defined version number of this message store.

For information about managing predefined message properties, see:

- "MessageStoreProperties class [QAnywhere C++]" on page 364
- "MessageProperties class [QAnywhere .NET]" on page 181
- "MessageStoreProperties interface [QAnywhere Java]" on page 476
- SQL API: "Message store properties" on page 647

Custom client message store properties

QAnywhere allows you to define your own client message store properties using the QAnywhere C++, Java, SQL or .NET APIs. These properties are shared between applications connected to the same

message store. They are also synchronized to the server message store so that they are available to server-side transmission rules for this client.

Client message store property names are case insensitive. You can use a sequence of letters, digits, and underscores, but the first character must be a letter. The following names are reserved and may not be used as message store property names:

- NULL
- TRUE
- FALSE
- NOT
- AND
- OR
- BETWEEN
- LIKE
- IN
- IS
- ESCAPE (SQL Anywhere message stores only)
- Any name beginning with **ias_**

Using custom client message store property attributes

Client message store properties can have attributes that you define. An attribute is defined by appending a dot after the property name followed by the attribute name. The main use of this feature is to be able to use information about your network in your transmission rules.

Limited support is provided for property attributes when using UltraLite as a client message store. UltraLite message stores only support the predefined ias_Network property.

Example (SQL Anywhere only)

The following is a simple example of how to set custom client message store property attributes. In this example, the Object property has two attributes: Shape and Color. The value of the Shape attribute is Round and the value of the Color attribute is Blue.

```
// C++ example.
mgr->setStringStoreProperty( "Object.Shape", "Round" );
mgr->setStringStoreProperty( "Object.Color", "Blue" );

// C# example.
mgr.SetStoreStringProperty( "Object.Shape", "Round" );
mgr.SetStringStoreProperty( "Object.Color", "Blue" );

// Java example
mgr.setStringStoreProperty( "Object.Shape", "Round" );
mgr.setStringStoreProperty( "Object.Color", "Blue" );

-- SQL example
BEGIN
    CALL ml_qa_setstoreproperty( 'Object.Shape', 'Round' );
    CALL ml_qa_setstoreproperty( 'Object.Color', 'Blue' );
    COMMIT;
END
```

All client message store properties have a Type attribute that initially has no value. The value of the Type attribute must be the name of another property. When setting the Type attribute of a property, the property inherits the attributes of the property being assigned to it. In the following example, the Object property inherits the attributes of the Circle property. Therefore, the value of Object.Shape is Round and the value of Object.Color is Blue.

```
// C++ example
QAManager qa_manager;
qa_manager->setStoreStringProperty( "Circle.Shape", "Round" );
qa_manager->setStoreStringProperty( "Circle.Color", "Blue" );
qa_manager->setStoreStringProperty( "Object.Type", "Circle" );

// C# example
QAManager qa_manager;
qa_manager.SetStringStoreProperty( "Circle.Shape", "Round" );
qa_manager.SetStringStoreProperty( "Circle.Color", "Blue" );
qa_manager.SetStringStoreProperty( "Object.Type", "Circle" );

// Java example
QAManager qa_manager;
qa_manager.setStringStoreProperty( "Circle.Shape", "Round" );
qa_manager.setStringStoreProperty( "Circle.Color", "Blue" );
qa_manager.setStringStoreProperty( "Object.Type", "Circle" );

-- SQL example
BEGIN
    CALL ml_qa_setstoreproperty( 'Circle.Shape', 'Round' );
    CALL ml_qa_setstoreproperty( 'Circle.Color', 'Blue' );
    CALL ml_qa_setstoreproperty( 'Object.Type', 'Circle');
    COMMIT;
END
```

Example

The following C# example shows how you can use message store properties to provide information about your network to your transmission rules.

Assume you have a Windows laptop that has the following network connectivity options: LAN, Wireless LAN, and Wireless WAN. Access to the network via LAN is provided by a network card named My LAN Card. Access to the network via Wireless LAN is provided by a network card named My Wireless LAN Card. Access to the network via Wireless WAN is provided by a network card named My Wireless WAN Card.

Assume you want to develop a messaging application that sends all messages to the server when connected using LAN or Wireless LAN and only high priority messages when connected using Wireless WAN. You define high priority messages as those whose priority is greater than or equal to 7.

First, find the names of your network adapters. The names of network adapters are fixed when the card is plugged in and the driver is installed. To find the name of a particular network card, connect to the network through that adapter, and then run qaagent with the -vn option. The QAnywhere Agent displays the network adapter name, as follows:

```
"Listener thread received message '[netstat] network-adapter-name !...'
```


Next, define three client message store properties for each of the network types: LAN, WLAN, and WWAN. Each of these properties are assigned a Cost attribute. The Cost attribute is a value between 1 and 3 and represents the cost incurred when using the network. A value of 1 represents the lowest cost.

```
QAManager qa_manager;
qa_manager.SetStoreProperty( "LAN.Cost", "1" );
qa_manager.SetStoreProperty( "WLAN.Cost", "2" );
qa_manager.SetStoreProperty( "WWAN.Cost", "3" );
```

Next, define three client message store properties, one for each network card that is used. The property name must match the network card name. Assign the appropriate network classification to each property by assigning the network type to the Type attribute. Each property therefore inherits the attributes of the network types assigned to them.

```
QAManager qa_manager;
qa_manager.SetStoreProperty( "My LAN Card.Type", "LAN" );
qa_manager.SetStoreProperty( "My Wireless LAN Card.Type", "WLAN" );
qa_manager.SetStoreProperty( "My Wireless WAN Card.Type", "WWAN" );
```

When network connectivity is established, QAnywhere automatically defines the Adapter attribute of the ias_Network property to one of My LAN Card, My Wireless LAN Card or My Wireless WAN Card, depending on the network in use. Similarly, it automatically sets the Type attribute of the ias_Network property to one of My LAN Card, My Wireless LAN Card, or My Wireless WAN Card so that the ias_Network property inherits the attributes of the network being used.

Finally, create the following transmission rule.

```
automatic=ias_Network.Cost < 3 or ias_Priority >= 7
```

For more information about transmission rules, see "QAnywhere transmission and delete rules" on page 729.

Enumeration of client message store properties

The QAnywhere .NET, C++, and Java APIs can provide an enumeration of predefined and custom client message store properties.

.NET example

See "QAManagerBase.GetStorePropertyNames method [QAnywhere .NET]".

```
// qaManager is a QAManager instance.
IEnumerator propertyNames = qaManager.GetStorePropertyNames();
```

C++ example

See "QAManagerBase.beginEnumStorePropertyNames method [QAnywhere C++]".

```
// qaManager is a QAManager instance.
qa_store_property_enum_handle handle = qaManager-
>beginEnumStorePropertyNames();
qa_char propertyName[256];

if( handle != qa_null ) {
    while(qaManager->nextStorePropertyName(handle, propertyName, 255) != -1)
```

```
{
        // Do something with the message store property name.
    }
    // Message store properties cannot be set after
    // the beginEnumStorePropertyNames call
    // and before the endEnumStorePropertyNames call.
    qaManager->endEnumStorePropertyNames(handle);
}
```

Java example

See "QAManagerBase.getStorePropertyNames method [QAnywhere Java]" on page 532.

```
// qaManager is a QAManager instance.
Enumeration propertyNames = qaManager.getStorePropertyNames();
```

Management of client message store properties in your application

The following QAManagerBase methods can be used to get and set client message store properties.

C++ methods to manage client message store properties

- qa_bool getBooleanStoreProperty(qa_const_string name, qa_bool * value)
- qa_bool setBooleanStoreProperty(qa_const_string name, qa_bool value)
- qa_bool getByteStoreProperty(qa_const_string name, qa_byte * value)
- qa_bool setByteStoreProperty(qa_const_string name, qa_byte value)
- qa_bool getShortStoreProperty(qa_const_string name, qa_short * value)
- qa_bool setShortStoreProperty(qa_const_string name, qa_short value)
- qa_bool getIntStoreProperty(qa_const_string name, qa_int * value)
- qa_bool setIntStoreProperty(qa_const_string name, qa_int value)
- qa_bool getLongStoreProperty(qa_const_string name, qa_long * value)
- qa_bool setLongStoreProperty(qa_const_string name, qa_long value)
- qa_bool getFloatStoreProperty(qa_const_string name, qa_float * value)
- qa_bool setFloatStoreProperty(qa_const_string name, qa_float value)
- qa_bool getDoubleStoreProperty(qa_const_string name, qa_double * value)
- qa_bool setDoubleStoreProperty(qa_const_string name, qa_double value)
- qa_int getStringStoreProperty(qa_const_string name, qa_string value, qa_int len)
- qa_bool setStringStoreProperty(qa_const_string name, qa_const_string value)
- qa_store_property_enum_handle QAManagerBase::beginEnumStorePropertyNames()
- virtual qa_int QAManagerBase::nextStorePropertyName(qa_store_property_enum_handle h, qa_string buffer, qa_int bufferLen)
- virtual void QAManagerBase::endEnumStorePropertyNames(qa_store_property_enum_handle h)

See "QAManagerBase class [QAnywhere C++]" on page 394.

C# methods to manage client message store properties

- Object GetStoreProperty(String name)
- void SetStoreProperty(String name, Object value)
- boolean GetBooleanStoreProperty(String name)
- void SetBooleanStoreProperty(String name, boolean value)
- byte GetByteStoreProperty(String name)
- void SetByteStoreProperty(String name, byte value)
- short GetShortStoreProperty(String name)
- void SetShortStoreProperty(String name, short value)
- int GetIntStoreProperty(String name)
- void SetIntStoreProperty(String name, int value)
- long GetLongStoreProperty(String name)
- void SetLongStoreProperty(String name, long value)
- float GetFloatStoreProperty(String name)
- void SetFloatStoreProperty(String name, float value)
- double GetDoubleStoreProperty(String name)
- void SetDoubleStoreProperty(String name, double value)
- String GetStringStoreProperty(String name)
- void SetStringStoreProperty(String name, String value)
- IEnumerator GetStorePropertyNames()

See "QAManagerBase interface [QAnywhere .NET]" on page 228.

Java methods to manage client message store properties

- boolean getBooleanStoreProperty(String name)
- void setBooleanStoreProperty(String name, boolean value)
- byte getByteStoreProperty(String name)
- void setByteStoreProperty(String name, byte value)
- double getDoubleStoreProperty(String name)
- void setDoubleStoreProperty(String name, double value)
- float getFloatStoreProperty(String name)
- void setFloatStoreProperty(String name, float value)
- int getIntStoreProperty(String name)
- void setIntStoreProperty(String name, int value)
- long getLongStoreProperty(String name)
- void setLongStoreProperty(String name, long value)
- short getShortStoreProperty(String name)
- void setShortStoreProperty(String name, short value)
- void setStringStoreProperty(String name, String value)
- String getStringStoreProperty(String name)
- java.util.Enumeration getStorePropertyNames()

See "QAManagerBase interface [QAnywhere Java]" on page 514.

SQL stored procedures to manage client message store properties

- ml_qa_getstoreproperty
- ml_qa_setstoreproperty

See "Message store properties" on page 647.

Server properties

You can set server properties in Sybase Central or with a server management request. Server properties are always stored in the database. See:

- "Server property setup with a server management request" on page 161
- "Setting server properties with Sybase Central" on page 726

Server properties

- **ianywhere.qa.server.autoRulesEvaluationPeriod** The time in milliseconds between evaluations of rules, including message transmission and persistence rules. Since, typically, rules are evaluated dynamically as messages are transmitted to the server store, the rule evaluation period is only for rules that are timing-sensitive. The default value is **60000** (one minute).

- **ianywhere.qa.server.compressionLevel** The default amount of compression applied to each message received by a QAnywhere connector. The compression is an integer between 0 and 9, with 0 being no compression and 9 being the highest compression. The default is **0**.

  If you also set the compression level for a connector in the connector properties file, this setting is overridden for that connector. See "JMS connector property configuration" on page 132.

- **ianywhere.qa.server.connectorPropertiesFiles**

  Note
  Replaced by Sybase Central.

  A list of one or more files that specify the configuration of QAnywhere connectors to an external message system such as JMS. The default is no connectors.

  See "Connectors" on page 129.

- **ianywhere.qa.server.disableNotifications** Set this to true to disable notification from the server about pending messages. This disables the processing on the server that is required to initiate notifications to clients when messages are waiting on the server for those clients. Set to true in any setup where notifications cannot be sent from the server, such as when firewall restrictions make notifications impossible. The default is false.

- **ianywhere.qa.server.logLevel** The logging level of the messaging. The property value may be one of 1, 2, 3, or 4. 1 indicates that only message errors are logged. 2 additionally causes warnings to be logged. 3 additionally causes informational messages to be logged. 4 additionally causes more verbose informational messages to be logged, including details about each QAnywhere message that is transmitted with the MobiLink server. The default is **2**.

  These logging messages are output to the MobiLink server messages window. If the mlsrv12 -o or -ot option was specified, the messages are output to the MobiLink server message log file.

- **ianywhere.qa.server.id** Specifies the agent portion of the address to which to send server management requests. If this property is not set, this value is ianywhere.server.
- **ianywhere.qa.server.password.e** Specifies the password for authenticating server management requests. If this property is not set, the password is QAnywhere.

  See "Server management requests" on page 147.
- **ianywhere.qa.server.scheduleDateFormat** Specifies the date format used for server-side transmission rules. By default, the date format is yyyy-MM-dd.

Letter	Date component	Example
y	year	1996
M	month in year	July
d	day in month	10

- **ianywhere.qa.server.scheduleTimeFormat** Specifies the time format used for server-side transmission rules. By default, the time format is HH:mm:ss.

Letter	Date component	Example
a	AM/PM marker	PM
H	hour in day, a value between 0 and 23	0
k	hour in day, a value between 1 and 24	24
K	hour in AM/PM, a value between 0 and 11	0
h	hour in AM/PM, a value between 1 and 12	12
m	minute in hour	30
s	second in minute	55

- **ianywhere.qa.server.transmissionRulesFile**

  Note
  Replaced by Sybase Central.

  A file used to specify rules for governing the transmission and persistence of messages. By default, there are no filters for messages, and messages are deleted when the final status of the message has been transmitted to the message originator.

Setting server properties with Sybase Central

Set server properties with Sybase Central

1. Start Sybase Central:

   Click **Start** » **Programs** » **SQL Anywhere 12** » **Administration Tools** » **Sybase Central**.

2. Click **Connections** » **Connect With QAnywhere 12**.

3. Specify an **ODBC Data Source Name** or **ODBC Data Source File**, and the **User ID** and **Password** if required. Click **OK**.

4. Under **Server Message Stores** in the left pane, click the name of the data source.

5. Click **File** » **Properties**.

JMS connector properties

The following properties are used to configure the JMS connectors:

- **ianywhere.connector.nativeConnection** The Java class that implements the connector. It is for QAnywhere internal use only, and should not be deleted or modified.

- **ianywhere.connector.id (deprecated)** An identifier that uniquely identifies the connector. The default is the value of the connector property ianywhere.connector.address.

- **ianywhere.connector.address** The connector address that a QAnywhere client should use to address the connector. This address is also used to prefix all logged error, warning, and informational messages appearing in the MobiLink server messages window for this connector.

  See "Sending a QAnywhere message to a JMS connector" on page 133.

  In Sybase Central, set this property in the **Connector Wizard** on the **Connector Names** page in the **Connector name** field.

- **ianywhere.connector.incoming.priority** The priority, expressed as an integer, assigned to all incoming messages. If the value is unspecified or negative, the default for that type of connector is used. In JMS, the default is to use the priority of the JMS message. In web services, the default is 4.

- **ianywhere.connector.incoming.retry.max** The maximum number of times the connector retries transferring a JMS message to a QAnywhere message store before giving up. After the maximum number of failed attempts, the JMS message is re-addressed to the ianywhere.connector.jms.deadMessageDestination property value. The default is -1, which means that the connector does not give up.

- **ianywhere.connector.incoming.ttl** The time-to-live, expressed as an integer, assigned to all incoming messages measured in milliseconds. If the value is unspecified or negative, the default for

that type of connector is used. If the value is 0, messages do not expire. In JMS, the default is calculated using the expiration time of the JMS message. In web services, the default is 0.

- **ianywhere.connector.outgoing.deadMessageAddress** The address that a message is sent to when it cannot be processed. For example, if a message contains a JMS address that is malformed or unknown, the message is marked as unreceivable and a copy of the message is sent to the dead message address.

  If no dead message address is specified, the message is marked as unreceivable but no copy of the message is sent.

  In Sybase Central, you can set this property in the **Connector Properties** window, **Properties** tab, by clicking **New**.

- **ianywhere.connector.logLevel** The amount of connector information displayed in the MobiLink server messages window and message log file. Values for the log level are as follows:

  - **1** Log error messages.
  - **2** Log error and warning messages.
  - **3** Log error, warning, and information messages.
  - **4** Log error, warning, information, and debug messages.

  In Sybase Central, set this property on the **Connector Properties** window, on the **General** tab, in the **Logging Level** section.

  You can also set this property for all connectors. To do this in Sybase Central, connect to a server message store, right-click the message store and click **Properties**. The **Properties** window allows you to set the following:

- **ianywhere.connector.compressionLevel** The default message compression factor of messages received from JMS: an integer between 0 and 9, with 0 indicating no compression and 9 indicating maximum compression.

  In Sybase Central, set this property on the **Connector Properties** window, on the **General** tab, in the **Compression Level** section.

  You can also set this property for all connectors. To do this in Sybase Central, connect to a server message store, click the task **Change Properties Of This Message Store**, and open the **Server Properties** tab.

- **ianywhere.connector.jms.deadMessageDestination** The address that a JMS message is sent to when it cannot be converted to a QAnywhere message. This might occur if the JMS message is an instance of an unsupported class, if the JMS message does not specify a QAnywhere address, if an unexpected JMS provider exception occurs, or if an unexpected QAnywhere exception occurs.

  In Sybase Central, set this property on the **Connector Properties** window, on the **JMS** tab, in the **Other** section, in the **Dead message destination** field.

- **ianywhere.connector.outgoing.retry.max** The default number of retries for messages going from QAnywhere to the external messaging system. The default value is 5. Specify 0 to have the connector retry forever.

  In Sybase Central, you can set this property in the **Connector Properties** window, **Properties** tab, by clicking **New**.

- **ianywhere.connector.runtimeError.retry.max** The number of times a connector retries a message that causes a RuntimeException. If a dead message queue is specified, the message is put in that queue. Otherwise, the message is marked as unreceivable and skipped. Specify a value of 0 to have the server never give up.

- **ianywhere.connector.startupType** Startup types can be automatic, manual, or disabled.

- **xjms.jndi.authName** The authentication name to connect to the external JMS JNDI name service.

  In Sybase Central, set this property in the **Connector Wizard**, **JNDI Settings** page, **User name** field; or on the **Connector Properties** window on the **JMS** tab, **JNDI** section, **User name** field.

- **xjms.jndi.factory** The factory name used to access the external JMS JNDI name service. In Sybase Central, set this property in the **Connector Wizard**, **JNDI Settings** page, **JNDI factory** field; or on the **Connector Properties** window on the **JMS** tab, **JNDI** section, **JNDI Factory** field,

- **xjms.jndi.password.e** The authentication password to connect to the external JMS JNDI name service.

  In Sybase Central, set this property in the **Connector Wizard**, **JNDI Settings** page, **Password** field; or on the **Connector Properties** window on the **JMS** tab, **JNDI** section, **Password** field.

- **xjms.jndi.url** The URL to access the JMS JNDI name service.

  In Sybase Central, set this property in the **Connector Wizard**, **JNDI Settings** page, **Name service URL** field; or on the **Connector Properties** window on the **JMS** tab, **JNDI** section, **URL** field.

- **xjms.password.e** The authentication password to connect to the external JMS provider.

- **xjms.queueConnectionAuthName** The user ID to connect to the external JMS queue connection.

  In Sybase Central, set this property in the **Connector Wizard**, **JMS Queue Settings** page, **User name** field; or on the **Connector Properties** window on the **JMS** tab, **Queue** section, **User name** field.

- **xjms.queueConnectionPassword.e** The password to connect to the external JMS queue connection.

  In Sybase Central, set this property in the **Connector Wizard**, **JMS Queue Settings** page, **Password** field; or on the **Connector Properties** window on the **JMS** tab, **Queue** section, **Password** field.

- **xjms.queueFactory** The external JMS provider queue factory name.

In Sybase Central, set this property in the **Connector Wizard**, **JMS Queue Settings** page, **Queue factory** field; or on the **Connector Properties** window on the **JMS** tab, **Queue** section, **Queue factory** field.

- **xjms.receiveDestination** The queue name used by the connector to listen for messages from JMS targeted for QAnywhere clients.

  In Sybase Central, set this property in the **Connector Wizard**, **Connector Names** page, **Receiver destination** field.

- **xjms.topicFactory** The external JMS provider topic factory name.

  In Sybase Central, set this property in the **Connector Wizard**, **JMS Topic Settings** page, **Topic Factory** field; or on the **Connector Properties** window on the **JMS** tab, **Topic** section, **Topic factory** field.

- **xjms.topicConnectionAuthName** The user ID to connect to the external JMS topic connection.

  In Sybase Central, set this property in the **Connector Wizard**, **JMS Topic Settings** page, **User name** field; or on the **Connector Properties** window on the **JMS** tab, **Topic** section, **User name** field.

- **xjms.topicConnectionPassword.e** The password to connect to the external JMS topic connection.

  In Sybase Central, set this property in the **Connector Wizard**, **JMS Topic Settings** page, **Password** field; or on the **Connector Properties** window on the **JMS** tab, **Topic** section, **Password** field.

QAnywhere transmission and delete rules

Rule syntax

Rule syntax

Each rule has the following form:

schedules=*condition*

Schedule syntax

Schedule syntax

schedules : { **AUTOMATIC** | *schedule-spec* ,... }

schedule-spec :
{ **START TIME** *start-time* | **BETWEEN** *start-time* **AND** *end-time* }
[**EVERY** *period* { **HOURS** | **MINUTES** | **SECONDS** }]
[**ON** { **(** *day-of-week*, ... **)** | **(** *day-of-month*, ... **)** }]
[**START DATE** *start-date*]

Parameters

- **AUTOMATIC** For transmission rules, rules are evaluated when a message changes state or there is a change in network status. For delete rules, messages that satisfy the delete rule condition are deleted when a message transmission is initiated.

- **schedule-spec** Schedule specifications other than AUTOMATIC specify times when conditions are to be evaluated. At those scheduled times, the corresponding condition is evaluated.

- **START TIME** The first scheduled time for each day on which the event is scheduled. If a START DATE is specified, the START TIME refers to that date. If no START DATE is specified, the START TIME is on the current day (unless the time has passed) and each subsequent day (if the schedule includes EVERY or ON).

- **BETWEEN ... AND ...** A range of times during the day outside which no scheduled times occur. If a START DATE is specified, the scheduled times do not occur until that date.

- **EVERY** An interval between successive scheduled events. Scheduled events occur only after the START TIME for the day, or in the range specified by BETWEEN ... AND.

- **ON** A list of days on which the scheduled events occur. The default is every day if EVERY is specified. Days can be specified as days of the week or days of the month.

  Days of the week are Mon, Tues, and so on. You may also use the full forms of the day, such as Monday. You must use the full forms of the day names if the language you are using is not English, is not the language requested by the client in the connection string, and is not the language that appears in the server messages window.

  Days of the month are integers from 0 to 31. A value of 0 represents the last day of any month.

- **START DATE** The date on which scheduled events are to start occurring. The default is the current date.

Usage

You can create more than one schedule for a given condition. This permits complex schedules to be implemented.

A schedule specification is recurring if its definition includes EVERY or ON; if neither of these reserved words is used, the schedule specifies at most a single time. An attempt to create a non-recurring schedule for which the start time has passed generates an error.

Each time a scheduled time occurs, the associated condition is evaluated and then the next scheduled time and date is calculated.

The next scheduled time is computed by inspecting the schedule or schedules, and finding the next schedule time that is in the future. If a schedule specifies every minute, and it takes 65 seconds to evaluate the conditions, it runs every two minutes. If you want execution to overlap, you must create more than one rule.

1. If the EVERY clause is used, find whether the next scheduled time falls on the current day, and is before the end of the BETWEEN ... AND range. If so, that is the next scheduled time.

2. If the next scheduled time does not fall on the current day, find the next date on which the event is to be executed.

3. Find the START TIME for that date, or the beginning of the BETWEEN ... AND range.

The QAnywhere schedule syntax is derived from the SQL Anywhere CREATE EVENT schedule syntax.

Keywords are case insensitive.

See also

- "CREATE EVENT statement" [*SQL Anywhere Server - SQL Reference*]

Example

The following sample server transmission rules file applies to the client identified by the client message store ID sample_store_id. It creates a dual schedule: high priority messages are sent once an hour. The schedule is `every 1 hours` and the condition is `ias_priority=9`. Also, between the hours of 8 A.M. and 9 A.M., high priority messages are sent every minute.

```
[sample_store_id]
; This rule governs when messages are transmitted to the client
; store with id sample_store_id.
;
   START TIME '06:00:00' EVERY 1 hours = ias_Priority = 9
   BETWEEN '08:00:00' AND '09:00:00' EVERY 1 minutes = ias_Priority = 9
```

Condition syntax

QAnywhere conditions use a SQL-like syntax. Conditions are evaluated against messages in the message store. A condition evaluates to true, false, or unknown. If a condition is empty, all messages are judged to satisfy the condition. Conditions can be used in transmission rules, delete rules, and the QAnywhere programming APIs.

Keywords and string comparisons are case insensitive.

Syntax

condition :
expression **IS** [**NOT**] **NULL**
| *expression compare expression*
| *expression* [**NOT**] **BETWEEN** *expression* **AND** *expression*
| *expression* [**NOT**] **LIKE** *pattern* [**ESCAPE** *character*]
| *expression* [**NOT**] **IN (** *string*, ... **)**
| **NOT** *condition*
| *condition* **AND** *condition*
| *condition* **OR** *condition*
| **(** *condition* **)**

compare: **= |> | < | >= | <= | <>**

expression:
constant
| *arithmetic-expression*

| *rule-variable*
| **(** *expression* **)**
| | *rule-function* **(** *expression*, ... **)**

constant: *integer* | *floating-point number* | *string* | *boolean*

arithmetic-expression:
numeric-constant
| *rule-variable*
| - *arithmetic-expression*
| *arithmetic-expression operator arithmetic-expression*
| *(arithmetic-expression)*
| *rule-function (expression, ...)*

numeric-constant: *integer* | *floating-point number*

integer: An integer in the range -2**63 to 2**63-1.

floating-point number: A number in scientific notation in the range 2.2250738585072e-308 to 1.79769313486231e+308.

string: A sequence of characters enclosed in single quotes. A single quote in a string is represented by two consecutive single quotes.

boolean: A statement that is **TRUE** or **FALSE**, **T** or **F**, **Y** or **N**, **1** or **0**.

operator: **+** | **-** | ***** | **/**

rule-variable:

See "Rule variables" on page 735.

rule-function:

See "Rule functions" on page 734.

Parameters

- **BETWEEN** The BETWEEN condition can evaluate as true, false, or unknown. Without the NOT keyword, the condition evaluates as true if *expression* is greater than or equal to the start expression and less than or equal to the end expression.

  The NOT keyword reverses the meaning of the condition but leaves UNKNOWN unchanged.

  The BETWEEN condition is equivalent to a combination of two inequalities:

  [**NOT**] **(** *expression* **>=** *start-expression* **AND** *arithmetic-expression* **<=** *end-expr* **)**

  For example:

  - `age BETWEEN 15 AND 19` is equivalent to `age >=15 AND age <= 19`
  - `age NOT BETWEEN 15 AND 19` is equivalent to `age < 15 OR age > 19`.

- **IN** The IN condition evaluates according to the following rules:

- True if *expression* is not null and equals at least one of the values in the list.
- Unknown if *expression* is null and the values list is not empty, or if at least one of the values is null and expression does not equal any of the other values.
- False if none of the values are null, and *expression* does not equal any of the values in the list.

The NOT keyword interchanges true and false.

For example:

- `Country IN ( 'UK', 'US', 'France' )` is true for `'UK'` and false for `'Peru'`. It is equivalent to the following:

```
( Country = 'UK' )       \
OR ( Country = 'US' )    \
OR ( Country = 'France' )
```

- `Country NOT IN ( 'UK', 'US', 'France' )` is false for `'UK'` and true for `'Peru'`. It is equivalent to the following:

```
NOT ( ( Country = 'UK' )         \
      OR ( Country = 'US' )      \
      OR ( Country = 'France' ) )
```

- **LIKE** The LIKE condition can evaluate as true, false, or unknown.

  Without the NOT keyword, the condition evaluates as true if *expression* matches the *like expression*. If either *expression* or *like expression* is null, this condition is unknown.

  The NOT keyword reverses the meaning of the condition, but leaves unknown unchanged.

  The *like expression* may contain any number of wildcards. The wildcards are:

Wildcard	Matches
_ (underscore)	Any one character
% (percent)	Any string of zero or more characters

  For example:

  - `phone LIKE 12%3` is true for `'123'` or `'12993'` and false for `'1234'`
  - `word LIKE 's_d'` is true for `'sad'` and false for `'said'`
  - `phone NOT LIKE '12%3'` is false for `'123'` or `'12993'` and true for `'1234'`

- **ESCAPE CHARACTER** The ESCAPE CHARACTER is a single character string literal whose character is used to escape the special meaning of the wildcard characters (**_**, **%**) in *pattern*. For example:

  - `underscored LIKE '\_%' ESCAPE '\'` is true for `'_myvar'` and false for `'myvar'`.

- **IS NULL** The IS NULL condition evaluates to true if the rule-variable is unknown; otherwise it evaluates to false. The NOT keyword reverses the meaning of the condition. This condition cannot evaluate to unknown.

Rule functions

You can use the following functions in transmission rules:

Syntax	Description
DATEADD(*datepart*, *count*, *datetime* **)**	Returns a datetime produced by adding several date parts to a datetime. The *datepart* can be one of year, quarter, month, week, day, hour, minute, or second. For example, the following example adds two months, resulting in the value 2006-07-02 00:00:00.0: `DATEADD( month, 2, '2006/05/02' )`
DATEPART(*datepart*, *date* **)**	Returns the value of part of a datetime value. The datepart can be one of year, quarter, month, week, day, dayofyear, weekday, hour, minute, or second. For example, the following example gets the month May as a number, resulting in the value 5: `DATEPART( month, '2006/05/02' )`
DATETIME(*string* **)**	Converts a string value to a datetime. The string must have the format 'yyyy-mm-dd hh:nn:ss'.
LENGTH(*string* **)**	Returns the number of characters in a string.
SUBSTRING(*string*, *start*, *length* **)**	Returns a substring of a string. The *start* is the start position of the substring to return, in characters. The *length* is the length of the substring to return, in characters.

Example

The following delete rule deletes all messages that entered a final state more than 10 days ago:

```
START TIME '06:00:00' every 1 hours = ias_Status >= ias_FinalState \
  AND ias_StatusTime < DATEADD( day, -10, ias_CurrentTimestamp) \
  AND ias_TransmissionStatus = ias_Transmitted
```


Rule variables

QAnywhere rule variables can be used in the condition part of rules. You can use the following as rule variables:

- "Message properties"
- "Client message store properties"
- "Variables defined by the rule engine"

Using properties as rule variables

Message properties and message store properties can be used as transmission rule variables. In both cases you can use predefined properties or you can create custom properties. If you have a message property and a message store property with the same name, the message property is used. To override this precedence, you can explicitly reference the property as follows:

- Preface a message store property name with ias_Store.
- Preface a message property name with ias_Message.

For example, the following automatic transmission rule selects all messages with the custom message property **urgent** set to **yes**:

```
automatic = ias_Message.urgent = 'yes'
```

The following automatic transmission rule selects messages when the custom message store property **transmitNow** is set to **yes**:

```
automatic = ias_Store.transmitNow = 'yes'
```

Variables defined by the rule engine

The following variables are defined by the rule engine:

- **ias_Address** The address of the message. For example, myclient\myqueue.
- **ias_ContentSize** The size of the message content. If the message is a text message, this is the number of characters. If the message is binary, this is the number of bytes.
- **ias_ContentType** The type of message:

IAS_TEXT_CONTENT	The message content consists of unicode characters.
IAS_BINARY_CONTENT	The message content is treated as an uninterpreted sequence of bytes.

- **ias_CurrentDate** The current date.

  A string can be compared against ias_currentDate if it is supplied in one of two ways:

- as a string of format, which is interpreted unambiguously.
- as a string according to the date_format database option set for the client message store database.

  See "Database options" [*SQL Anywhere Server - Database Administration*] and "date_format option" [*SQL Anywhere Server - Database Administration*].

- **ias_CurrentTime** The current time.

  A string can be compared against ias_CurrentTime if the hours, minutes, and seconds are separated by colons and a period in the format hh:mm:ss.sss. A 24-hour clock is assumed unless **am** or **pm** is specified. See "time_format option" [*SQL Anywhere Server - Database Administration*].

- **ias_CurrentTimestamp** The current timestamp (current date and time). See "time_format option" [*SQL Anywhere Server - Database Administration*].

- **ias_Expires** The date and time when the message expires if it is not delivered.

- **ias_Network** Information about the current network in use. ias_Network is a special transmission variable. It has many built-in attributes that provide information regarding the current network that is being used by the device.

- **ias_Priority** The priority of message: an integer between 0 and 9, where 0 indicates less priority and 9 indicates more priority.

- **ias_Status** The current status of the message. The values can be:

IAS_CANCEL-LED_STATE	The message has been canceled.
IAS_EXPIRED_STATE	The message expired before it could be received by the intended recipient.
IAS_FINAL_STATE	The message is received or expired. Therefore, `>=IAS_FINAL_STATE` means that the message is received or expired, and `<IAS_FINAL_STATE` means that the message is neither received nor expired.
IAS_PEND-ING_STATE	The message has not yet been received by the intended recipient.
IAS_RE-CEIVED_STATE	The message was received by the intended recipient.
IAS_UNRECEIVA-BLE_STATE	The message has been marked as unreceivable because it is either malformed or there were too many failed attempts to deliver it.

See also:

- "MessageProperties.STATUS field [QAnywhere .NET]" on page 188
- "MessageProperties.STATUS variable [QAnywhere C++]" on page 363
- "MessageProperties.STATUS variable [QAnywhere Java]" on page 475

- **ias_StatusTime** The time at which the message became its current status.

  See also:

  - "MessageProperties.STATUS_TIME field [QAnywhere .NET]" on page 189
  - "MessageProperties.STATUS_TIME variable [QAnywhere C++]" on page 363
  - "MessageProperties.STATUS_TIME variable [QAnywhere Java]" on page 476

- **ias_TransmissionStatus** The synchronization status of the message. It can be one of:

IAS_UNTRANSMITTED	The message has not been transmitted to its intended recipient message store.
IAS_TRANSMITTED	The message has been transmitted to its intended recipient message store.
IAS_DO_NOT_TRANSMIT	The recipient and originating message stores are the same so no transmission is necessary.
IAS_TRANSMITTING	The message has been transmitted to its intended recipient, but that transmission has yet to be confirmed. There is a possibility that the message transmission was interrupted, and that QAnywhere may transmit the message again.

  See also:

  - "MessageProperties.TRANSMISSION_STATUS field [QAnywhere .NET]" on page 189
  - "MessageProperties.TRANSMISSION_STATUS variable [QAnywhere C++]" on page 363
  - "MessageProperties.TRANSMISSION_STATUS variable [QAnywhere Java]" on page 476

See also

- For an example of how to create client store properties and use them in transmission rules, see "Using custom client message store property attributes" on page 719

Message transmission rules

You can specify transmission rules on the server and on the client.

See also

- "Client transmission rules" on page 738
- "Server transmission rules" on page 739

Client transmission rules

Client transmission rules govern the behavior of messages going from the client to the server. Client transmission rules are handled by the QAnywhere Agent.

By default, the QAnywhere Agent uses the automatic policy. You can change and customize this behavior by specifying a transmission rules file as the transmission policy for the QAnywhere Agent.

The following partial qaagent command line shows how to specify a rules file for the QAnywhere Agent:

```
qaagent -policy myrules.txt ...
```

The transmission rules file holds the following kinds of entry:

- **Rules** No more than one rule can be entered per line.

  Each rule must be entered on a single line, but you can use \ as a line continuation character.

- **Comments** Comments are indicated by a line beginning with either a **#** or **;** character. Any characters on that line are ignored.

You can also use transmission rules files to determine when messages are to be deleted from the message stores.

You can also use the Sybase Central QAnywhere 12 plug-in to create a QAnywhere Agent rules file.

See also

- “Rule syntax” on page 729
- “Condition syntax” on page 731
- “Determining when message transmission should occur on the client” on page 46
- “-policy qaagent option” on page 685
- “Client delete rules” on page 741
- “Message delete rules” on page 741

Example

For example, the following client transmission rules file specifies that during business hours only small, high priority messages should be transmitted, and any message can be transmitted outside business hours. This rule is automatic, which indicates that if the condition is satisfied, the message is transmitted immediately. This example demonstrates that conditions can use information derived from the message and other information such as the current time.

```
automatic=(ias_ContentSize < 100000 AND ias_Priority > 7 ) \
 OR datepart(Weekday,ias_CurrentDate) in ( 1, 7 ) \
 OR ias_CurrentTime < datetime('8:00:00') \
 OR ias_CurrentTime > datetime('18:00:00')
```


Server transmission rules

Setting default rules

You can specify server transmission rules for a particular message store or destination alias, or you can set default rules for all clients. Every user that does not have an explicit transmission rule uses the default rule.

To set default rules, you use the special client name ianywhere.server.defaultClient.

Scheduled server transmission rules

Keep the following points in mind when specifying scheduled server transmission rules:

- Automatic rules for a given client are evaluated whenever that client synchronizes and at the automatic rule evaluation period.
- Scheduled rules for a given client are evaluated on the specified schedule.
- The evaluation of a rule causes push notifications to be sent to clients that currently have messages satisfying the rule conditions.
- Every time a client synchronizes, messages that satisfy conditions of automatic rules for that client are transmitted to the client.
- If and only if a scheduled rule has caused a push notification to be sent to a client since the last time that client synchronized, all messages satisfying the condition of the scheduled rule at the time of the next synchronization are transmitted to the client.

Server transmission rule specification with a transmission rules file (deprecated)

You can create a server transmission rules file and specify it with the ianywhere.qa.server.transmissionRulesFile property in your QAnywhere messaging properties file.

To specify transmission rules for a particular client, precede a section of rules with the client message store ID in square brackets.

Default server transmission rules can be created that apply to all users.

To specify default transmission rules, start a section with the following line:

```
[ianywhere.server.defaultClient]
```

For new transmission rules to take effect, you must restart the MobiLink server. This only applies to transmission rules specified in a transmission rules file. Server transmission rules specified using Sybase Central or a server management request take effect immediately.

Example

The following section of a server transmission rules file creates the default rule that only high priority messages should be sent:

```
[ianywhere.server.defaultClient]
auto = ias_Priority > 6
```

In the following sample server transmission rules file, the rules apply only to the client identified by the client message store ID sample_store_id.

```
[sample_store_id]
; This rule governs when messages are transmitted to the client
; store with id sample_store_id.
;
;    ias_Priority >= 7
;
; Messages with priority 7 or greater should always be
; transmitted.
;
;    ias_ContentSize < 100
;
; Small messages (messages less than 100 characters or
; bytes in size) should always be transmitted.
;
;    ias_CurrentTime < '8:00am' OR ias_CurrentTime > '6:00pm'
;
; Messages outside business hours should always be
; transmitted

auto = ias_Priority >= 7 OR ias_ContentSize < 100 \
    OR ias_CurrentTime < datetime('8:00:00') \
    OR ias_CurrentTime > datetime('18:00:00')
```

In the following example, the rules apply only to the client identified by the client message store ID qanywhere.

```
[qanywhere]
; This rule governs when messages are transmitted to the client
; store with id qanywhere.
;
;    tm_Subject not like '%non-business%'
;
; Messages with the property tm_Subject set to a value that
; includes the phrase 'non-business' should not be transmitted
;
;    ias_CurrentTime < '8:00:00' OR ias_CurrentTime > '18:00:00'
;
; Messages outside business hours should always be
; transmitted

auto = tm_Subject NOT LIKE '%non-business%' \
    OR ias_CurrentTime < datetime('8:00am') OR ias_CurrentTime >
datetime('6:00pm')
```

See also

- For more information about the messaging properties file, see "-m mlsrv12 option" [*MobiLink - Server Administration*]
- For information about server delete rules, see "Server delete rules" on page 741

Message delete rules

Delete rules determine the persistence of messages in the client message store and the server message store.

Default behavior

A QAnywhere message expires when the expiry time has passed and the message has not been received or transmitted anywhere. After a message expires, it is deleted by the default delete rules. If a message has been received at least once, but not acknowledged, it is possible to receive it again, even if the expiry time passes.

Client delete rules

By default, messages are deleted from the client message store when the status of the message is determined to be received, expired, canceled, or undeliverable and the final state has been transmitted to the server message store. You may want messages to be deleted faster than that, or to hold on to messages longer. You do that by creating a delete section in your client transmission rules file. The delete section must be prefaced by **[system:delete]**.

The following is an example of the delete rules section in a client transmission rules file:

```
[system:delete]

; This rule governs when messages are deleted from the client
; store.
;
;    start time '1:00:00' on ( 'Sunday' )
;
; Messages are deleted every Sunday at 1:00 A.M.
;
;    ias_Status >= ias_FinalState
;
; Typically, messages are deleted when they reach a final
; state: received, unreceivable, expired, or canceled.

START TIME '1:00:00' ON ( 'Sunday' ) = ias_Status >= ias_FinalState
```

Server delete rules

By default, messages are deleted from the server message store when the status of the message is determined to be received, expired, canceled, or undeliverable and the final state has been transmitted back to the message originator. You may want to keep messages longer for purposes such as auditing.

Server-side delete rules apply to all messages in the server message store.

See also

- “Client transmission rules” on page 738
- “Server transmission rules” on page 739
- “AcknowledgementMode enumeration [QAnywhere .NET]” on page 302
- “AcknowledgementMode class [QAnywhere C++]” on page 354
- “AcknowledgementMode interface [QAnywhere Java]” on page 468
- For an explanation of ias_Status, see “Rule variables” on page 735

Index

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X

