U1 TECHNOLOGIES

AmbrosiaMQ™

An Event Management System

April 2009

Administrator's Guide

AmbrosiaMQ™

Documentation Revision: 042409

U1 Technologies
204 Tiburon Boulevard
San Rafael, CA 94901

phone: (415) 480-0318
fax: (415) 704-3275
web: http://www.u1.com
e-mail: info@u1.com

Contents

Preface

About this Manual

This manual, the *Administrator's Guide,* discusses system configuration and maintenance of AmbrosiaMQ Event Management System. Use this manual for instructions on installation procedure, message broker configuration, guaranteed delivery log file administration, and the use of various tools.

Documentation Set

The AmbrosiaMQ *Concepts and Capabilities Guide* introduces the concepts underlying an event management system. A comprehensive treatment of these concepts provides a useful background for systems analysts, developers, and project managers who wish to learn how to leverage AmbrosiaMQ to develop distributed business applications. The concepts discussed in that guide provide an essential groundwork for application architects and developers who will use these key concepts in the design and development of AmbrosiaMQ business applications. This conceptual groundwork leads naturally into the use of AmbrosiaMQ for building secure globally deployed Internet/Intranet enterprise class applications.

Other documentation for the software includes the Javadoc generated from the com.u1.client package as well as the directory of sample programs with their associated readme files. We encourage you to review these samples before writing your own applications

How to Use this Manual

Use this manual to install and configure AmbrosiaMQ on your system:

- "Chapter 1: Installation," provides detailed instructions for the initial installation of AmbrosiaMQ.
- "Chapter 2: Message Broker Configuration," describes the system administrator's options to change the default settings of the message broker. Read this chapter to learn how to change AmbrosiaMQ's port number, determine the location of log files, and establish detailed control over the guaranteed delivery log file usage.
- "Chapter 3: Interbroker Configuration," describes how to deploy and configure multiple AmbrosiaMQ brokers into collaborating clusters that are known as interbroker networks.
- "Chapter 4: System Messages," lists broker and client system messages

- "Appendix A: Glossary," contains brief explanations of the terms used in the manual. While we have attempted to use descriptive or industry terms, this creates a common lexicon.

Feedback Welcome

Your comments are valuable to us. Your opinions represent the most important input that we receive for the next generation of our documentation. We want to know what we can do to improve this manual. You can send your comments by electronic mail to us at:

support@u1.com

Or, you can mail your comments to:

AmbrosiaMQ Technical Publications Feedback
U1 Technologies
204 Tiburon Boulevard
San Rafael, CA 94901

Naming Conventions

U1CG denotes Interfaces with the prefix character `I` and Exceptions with the character `E`. In general, U1CG adheres to standard Java naming conventions for classes and methods.

Documentation Conventions

Documentation Conventions

The manual uses the following general conventions:

- Italics often designate a reference to other information. Used alone, italics refer to the title of another manual; for example:

  Refer to the *Administrator's Guide* for additional information.

- References to another chapter within this manual start with the chapter number and appear in quotations; for example:

  "Chapter 1: Installation," includes the detailed procedures for installing AmbrosiaMQ.

- References to other sections within a chapter use quotations; for example:

  Refer to "Running the Message Broker as a Windows Service," below, for more information.

Typographical Conventions

The manual uses the following typographical conventions:

- **Bold** emphasis on regular type represents a "button" on a graphical user interface window that performs an action; for example:

  Click **Subjects** to invoke the *Selector* window:

- When describing GUI windows, quotation marks enclosing regular text indicate displayed text; for example:

  "Members" lists all current group members.

  In the above example, "bill," "madeline," and "marshall" are group members of the group "BillsCo."

- Angle brackets (<>) enclose names of keyboard keys or required parameters that are user-specific; for example:

  Enter the name of the new configuration file and press <Enter>. Replace "<key>" with your license key.

- Within code examples, angle brackets enclose required actions that are user specific; for example:

  <prepare other RMs>

- Terms with their initial letter capitalized indicate Java objects, and appended closed parentheses indicate Java method calls; for example:

  When you publish a Message, it must have a subject. Use `Session.subscribe()` on your `Session` object.

- Courier font represents AmbrosiaMQ and Java classes, methods, arguments, and exceptions; for example:

  To allow subscription to remain in effect, the boolean parameter `forceUnsubscribeAll` of the method `disconnect()` needs to be set to `true`.

1
Installation

This chapter covers the installation of the AmbrosiaMQ Event Management System on various platforms.

Supported Platforms

AmbrosiaMQ version 4.5 is supported on the following platforms:

- Window 2000 and Windows XP
- Solaris 8
- Linux

System Requirements

The following system requirements are necessary for the installation of the AmbrosiaMQ Event Management System:

- 400 Megahertz CPU or greater
- 100 Megabytes of disk space
- 512 Megabytes of RAM

AmbrosiaMQ will support and take advantage of multiple CPUs within a system. Additionally, multiple network interface adapters may be used by the AmbrosiaMQ system.

Installed Components

The AmbrosiaMQ installer will install the following items on the target system:

- AmbrosiaMQ broker libraries
- AmbrosiaMQ client libraries
- Scripts or batch files for initializing, starting, and stopping the broker
- Java Runtime Environment version 1.5.0.-03
- Derby database support files
- AmbrosiaMQ configuration files

All files, including the JRE are deployed to the target directory or a sub directory of the target directory. No files are installed to system directories.

Optionally, the installer can create and install an auto-start script for the AmbrosiaMQ broker. If this option is selected, then the installer must be run as root to deploy the auto-start script into the system startup scheme. No other system configuration files are created or modified by the AmbrosiaMQ installer.

On Windows platforms, the AmbrosiaMQ installer creates uninstall registry keys.

Installation - GUI

The AmbrosiaMQ installation tool supports a GUI installer for the following platforms: Windows 2000, Windows XP, Solaris 8, and Linux.

Note: If your system does not have a graphical display, the AmbrosiaMQ installer supports a command line console mode. Please refer to the next section for details.

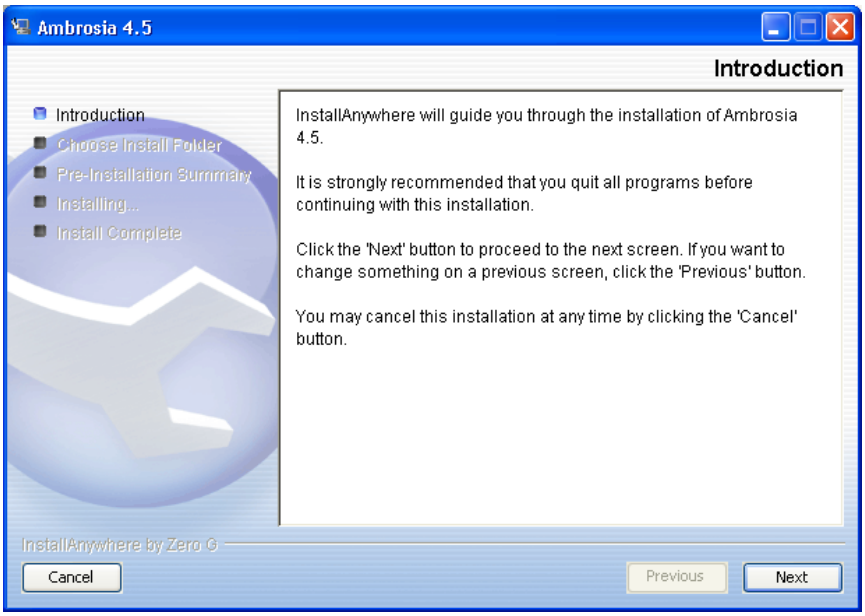
To begin the installation process, execute the "*setup"* binary executable provided with the AmbrosiaMQ distribution (from download or CD). The *"setup"* executable filename is platform dependent.

- Windows – setup.exe
- Solaris – setup.bin
- Linux – setup.bin

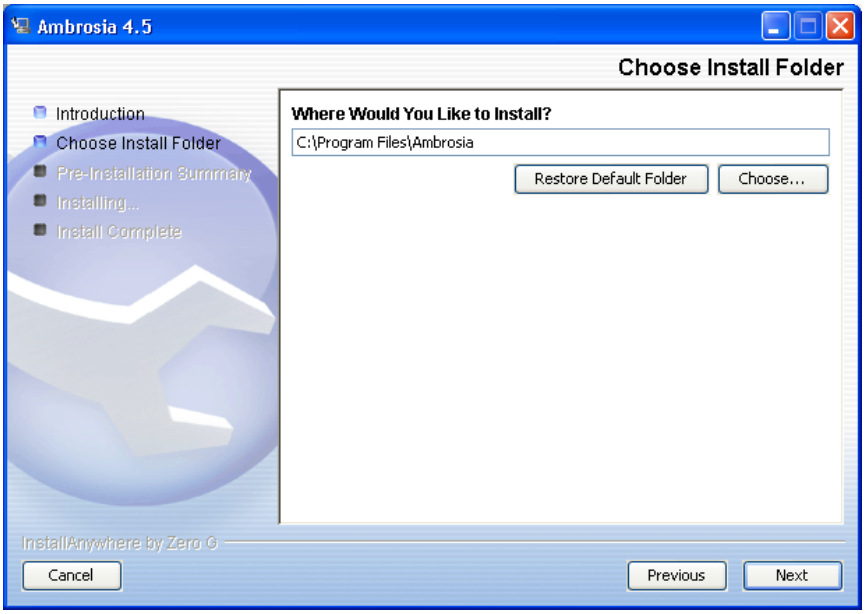
Note: If you downloaded the AmbrosiaMQ distribution, you may need to set execute permissions for the "*setup"* executable as follows:

```
chmod +x setup.bin
```

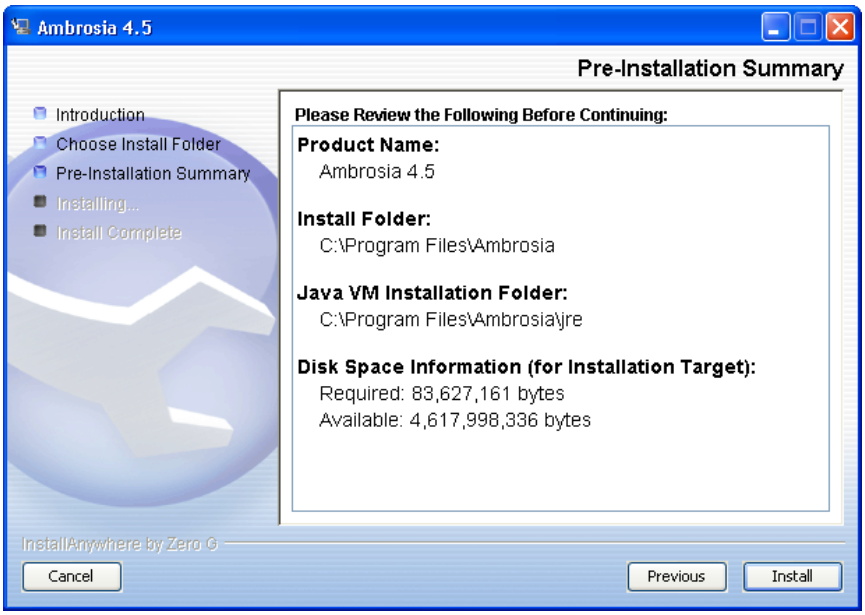
Once the *"setup"* executable has been started, the following dialog will be displayed:



After reviewing the introduction, clicking **Next** will allow you to choose your installation target directory.

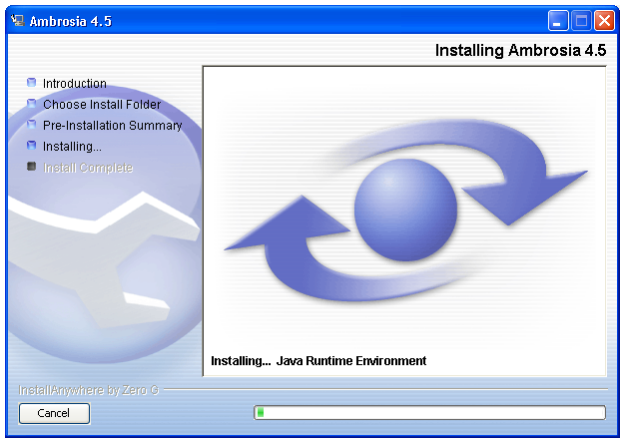


The installation folder should be a writable directory with at least 100 Megabytes of disk space. If you specify a directory that does not exist, the installer will automatically create the new directory. If you select a directory with an existing AmbrosiaMQ installation, then the installer will overwrite the existing installation. Clicking the **Next** button will cause the installer to display the pre-installation summary.

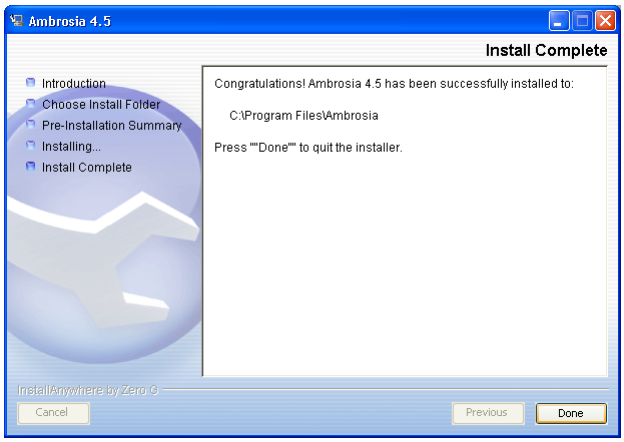


After reviewing the pre-installation summary, click **Next** to begin the installation deployment process. If you would like to change the installation target directory, click the **Previous** button to return the *Choose Installation Folder* step.

Once the installation deployment process begins, the installer will display a status dialog that contains a progress status bar.



Once the software has been deployed, an *Installation Complete* dialog will be displayed indicating the installation status (success or failure). Clicking the **Done** button will terminate the installer process. At this point, you are ready to start the AmbrosiaMQ message broker.



Installation – Console Mode

In addition to the GUI installer, the Unix platforms — Solaris and Linux, support a console mode installer. This may be useful on server systems that do not have a graphical display. To invoke the installer in console mode, run the following command:

```
./setup.bin –i console
```

Client Installation

The AmbrosiaMQ client deployment consists of libraries for communicating with the AmbrosiaMQ message broker for the supported languages. For Java clients, the AmbrosiaMQ client library is distributed in a file called: client.jar.

For C clients, the AmbrosiaMQ client library is distributed in a library file called amclient.so for Unix platforms, and amclient.dll for Windows platforms. All AmbrosiaMQ C clients also require the AmbrosiaMQ Java client library: client.jar.

In addition, all AmbrosiaMQ clients, Java or C, require a Java virtual machine of version 1.4.2 or higher (which includes Java 1.5 versions).

There is not an installation utility for installing the AmbrosiaMQ client package. This is because the AmbrosiaMQ client is typically a component of a third party application.

Installation Log

This installer will create an installation log named *AmbrosiaMQ_4.5_InstallLog*. This file contains a log of all files deployed including their installed location, all registry keys created or modified (Windows platforms), and all files modified by the installation process. The installation log is very useful in diagnosing and correcting installation issues.

Testing the Installation

Once the installation is complete, you can test starting the broker using the following process:

1. From a console window, change the current directory to the AmbrosiaMQ home directory (installation target directory). For example:

```
cd c:\AmbrosiaMQ
```

2. Initialize Broker Database

   On Windows run:

   ```
   windows\initdb
   ```

   On Solaris run:

   ```
   ./solaris/initdb
   ```

   On Linux run:

   ```
   ./linux/initdb
   ```

   The console should display output similar to the following:

```
C:\AmbrosiaMQ\windows>initdb
C\Tools\proja>java com.u1.broker.InitBrokerDatabase create
[2005-07-05 19:27:34] JDBC Driver:
org.apache.derby.jdbc.EmbeddedDriver
[2005-07-05 19:27:34] JDBC URL:
jdbc:derby:AmbrosiaMQ_db;create=true
[2005-07-05 19:27:34] JDBC Driver Version: 10.0
[2005-07-05 19:27:34] JDBC Compliant: false
[2005-07-05 19:27:34] JDBC Props: config/derby.cfg
[2005-07-05 19:27:34] Database User: user1

Database objects were removed.
Creating table MessageInfo.
Creating index MsgInfoIdx.
Creating table Messages.
Creating index MsgIdx.
Creating table UserIdMapping.
Creating index UidIdx.
Creating table Subscriptions.
Creating index SubIdx.
Creating table UndeliveredMessages.
Creating UndeliveredMessages indices.
Warning: UndeliveredMessages indices not created.
Creating table PreparedTxns.
Creating table TxnMessages.
Creating table SyncPoint.
Creating table BrokerMode.
Creating table PBMsgInfo.
Creating table PBMsgs.
Creating table Topology.
Creating table BrokerConnS.
Creating table CbrSubscriptions.
Creating table MFR.
Setting security mode
Setting sync point
Broker Database initialization is complete.

InitBrokerDatabase terminates.
```

3. Start the Broker

On Windows run:

```
windows\runbroker
```

On Solaris run:

```
./solaris/runbroker
```

On Linux run:

```
./linux/runbroker
```

The console should display output similar to the following:

```
C:\Tools\proja>java com.u1.broker.Broker
AmbrosiaMQ Broker: Version 4.5 on 06/30/05 A Protocol P20
Copyright (c) 2003 U1 Consulting Group, Inc. All Rights Reserved.

This version of AmbrosiaMQ will support up to 999999 concurrent
connections.
Security is disabled.


[2005-07-05 19:41:46] Broker: master
[2005-07-05 19:41:46] JRE Version: Sun Microsystems Inc. 1.5.0_03
C:\tools\proja
\jre
[2005-07-05 19:41:46] JRE Max Memory: 66650112

[2005-07-05 19:41:46] Current Directory: C:\Tools\proja\.

[2005-07-05 19:41:46] Class Path:
c:\tools\proja\lib\AmbrosiaMQ_server.jar;C:\tool
s\proja\lib\AmbrosiaMQ_client.jar;C:\tools\proja\lib\derby.jar;C:\tools\proj
a\lib\
atest.jar;C:\tools\proja\lib\tests.jar;.

[2005-07-05 19:41:46] Library Path:

[2005-07-05 19:41:46] OS: Windows XP x86 5.1
[2005-07-05 19:41:46] OS User: tom
[2005-07-05 19:41:46] Num Processors: 1

[2005-07-05 19:41:47] JDBC Driver:
org.apache.derby.jdbc.EmbeddedDriver
[2005-07-05 19:41:47] JDBC URL:
jdbc:derby:AmbrosiaMQ_db;create=true
[2005-07-05 19:41:47] JDBC Driver Version: 10.0
[2005-07-05 19:41:47] JDBC Compliant: false
[2005-07-05 19:41:47] JDBC Props: config/derby.cfg
[2005-07-05 19:41:47] Database User: user1

[2005-07-05 19:41:49] Java Block File Support Enabled - Block Size: 4096
[2005-07-05 19:42:04] Requested Acceptors: 8001,ssl://8002
tcp Acceptor on port 8001 now accepting...
ssl Acceptor on port 8002 now accepting...
```

Note: If the AmbrosiaMQ broker fails to start with the following message:

[2005-07-05 19:48:45] [146]
com.u1.broker.EStartupFailure: Broker already running

AmbrosiaMQ Message Broker exiting.

This error means there is a process already listening on the default broker acceptor ports 8001 or 8002. To correct this issue, either shutdown the conflicting process or reconfigure the AmbrosiaMQ broker to listen on different ports that are not in use.

4. Run a simple messaging test. This requires two console windows, one for the subscriber and one for the publisher. The current directory of the console windows should be the AmbrosiaMQ home directory. In separate console windows run the following commands.

First, run the subscriber application:

On Windows run:

```
windows\sub
```

On Solaris run:

```
./solaris/sub
```

On Linux run:

```
./linux/sub
```

Second, run the publisher application:

On Windows run:

```
windows\pub
```

On Solaris run:

```
./solaris/pub
```

On Linux run:

```
./linux/pub
```

After the publisher application has completed running, the subscriber console window should be similar to the following display output:

C:\AmbrosiaMQ>java SubscribeToSubject
Subscribing to subject: #
Message:
$SYS.broker.connect.a.MessageSender_3455240477991753551:
java.io.EOFException

Message: testSubject: This is a test message
Message:
$SYS.broker.drop.a.MessageSender_3455240477991753551:
java.io.EOFException

2

Broker Configuration

This chapter describes how to configure an AmbrosiaMQ message broker through the configuration file *ambroker.ini.*

Broker Configuration File (ambroker.ini)

The ambroker.ini file is the main repository for the AmbrosiaMQ broker properties. The ambroker.ini file must be located in the current directory for the AmbrosiaMQ broker process. The AmbrosiaMQ broker process must have read permissions for the ambroker.ini configuration files; it does not require write permissions.

General Broker Properties

Property Name	Default Value	Description
BROKER_NAME	None	Name of the broker. Should be 11 characters or less.
BROKER_PASSWORD	None	Must be set if INTERBROKER=true
LICENSE_KEY	None	Obtain license key from U1 Consulting Group, Inc.
ERROR_MSG_DIR	../messages	Location of error message catalogs for the broker.
ENABLE_SECURITY	False	Determines whether user authentication is required to connect to the broker and whether ACL policies are enforced by a broker.
ENABLE_MSG_EXPIRATION_NOTIFICATION	False	Determines whether the broker will publish an expired message notification when dropping an expired message.

Acceptor and Protocol Properties

Property Name	Default Value	Description
ACCEPTORS	8001, ssl://8002	Defines a list of broker protocol listeners (acceptors). Each acceptor is of the format <protocol>://<bindAddress>:<port>. If multiple acceptors are defined, then they should be separated by a comma. The <protocol> and <bindAddrss> are optional.
MAX_CONNECTIONS	1,000,000 concurrent connections	The maximum number of concurrent client connections that a broker will allow.
SOCKET_CLOSE_DELAY_MILLIS	10,000 milliseconds	The time to wait for buffered data to be sent before closing a socket.
SSL_CIPHERS	SSL_RSA_WITH_RC4_128_SHA	A comma separated list of SSL cipher suites to available for SSL connections on this broker. See Available SSL Cipher table for values.

The following SSL ciphers are available for use by the AmbrosiaMQ broker:

Available SSL Ciphers
SSL_RSA_WITH_RC4_128_SHA
SSL_RSA_WITH_RC4_128_MD5
TLS_RSA_WITH_AES_128_CBC_SHA
TLS_DHE_RSA_WITH_AES_128_CBC_SHA
TLS_DHE_DSS_WITH_AES_128_CBC_SHA
SSL_RSA_WITH_3DES_EDE_CBC_SHA
SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA
SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA
SSL_RSA_WITH_DES_CBC_SHA
SSL_DHE_RSA_WITH_DES_CBC_SHA
SSL_DHE_DSS_WITH_DES_CBC_SHA
TLS_RSA_WITH_AES_256_CBC_SHA

TLS_DHE_RSA_WITH_AES_256_CBC_SHA

Queue and Flow Control Properties

Property Name	Default Value	Description
INPUT_BUFFER_SIZE	None	Default size in bytes for the broker's BufferedInputStreams.
OUTPUT_BUFFER_SIZE	1024 bytes	Size in bytes of the broker's BufferedOutputStreams
OUTPUT_QUEUE_SIZE	1,000,000 bytes	Maximum size of an individual outgoing message queue on a broker.
OUTPUT_QUEUE_POOL_SIZE	10,000,000 bytes	Maximum total size of all outgoing messages queues on a broker.
FLOW_CONTROL_RESTART_ THRESHOLD	1000 bytes	Number bytes that must be available in a queue before sending restart notification.
QUEUE_ACTIVATION_ THRESHOLD	4 messages	The number of messages on a queue before it is considered active for sizing calculations.
QUEUE_ACTIVATION_ DELAY_MILLIS	3 seconds	How long a queue must remain above the activation threshold to be considered active for sizing calculations.
QUEUE_DEACTIVATION_ POLL_INTERVAL_MILLIS	30 seconds	How often to check a queue to see if it has become inactive for sizing calculations
SAVE_QUEUE_SIZE	500,000 bytes	Maximum size of the queue for persisting messages to the database.

Guaranteed Message Properties

Property Name	Default Value	Description
LOG_PATH	Current directory "."	Directory that contains the guaranteed message log files. This directory should always be located on a local disk.
LOG_QUEUE_SIZE	500,000 bytes	Maximum size of the log manager's queue.
MAX_LOG_FILE_SIZE	10,000,000 bytes	Maximum size of the guaranteed message log files.
SYNCPOINT_INTERVAL	1,000,000 bytes	Number of bytes written to the log before the broker's state is synced to disk and the database.
RABF_BLOCK_SIZE	8192 bytes	The minimum number of bytes to write and commit to disk in a single operation (block). Only used for pure Java implementation of the broker.

Database Properties

Property Name	Default Value	Description
DB_USER	None	The username to use when establishing a JDBC connection to the database.
DB_PASSWORD	None	The password to use when establishing a JDBC connection to the database.
DB_CONNECT	None	The JDBC connection string.
JDBC_DRIVER	None	The class name of the JDBC driver to use for making connections to the database.
DB_PROPERTIES	None	The name of a configuration file containing the database type mappings and other database vendor specific properties.
DB_MSG_CLEAN_THRESHOLD	5,000 messages	The number of unreferenced messages to retain in the database before performing a cleanup operation.
EXP_MSG_CLEAN_ THRESHOLD	10,000 messages	The number of expired messages to retain in the database before performing a cleanup operation.

3

Interbroker Configuration

This chapter describes how to deploy and configure multiple AmbrosiaMQ brokers into collaborating clusters that are known as interbroker networks.

Interbroker Overview

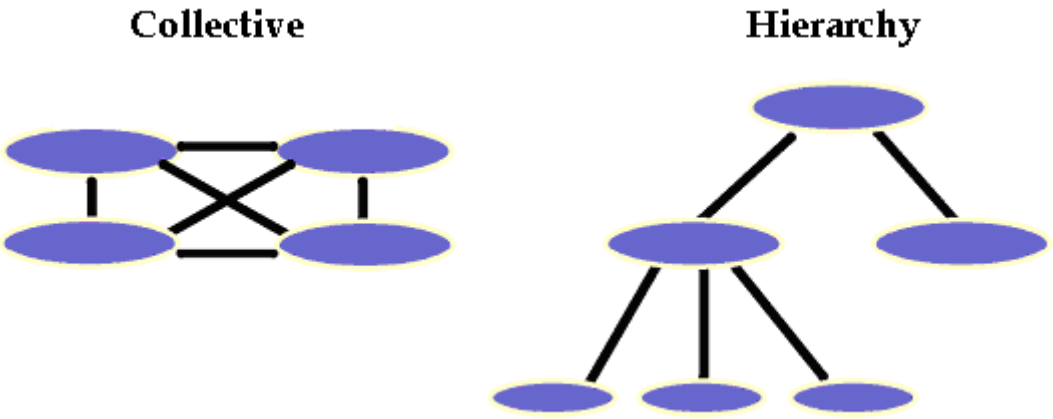
The interbroker network allows multiple AmbrosiaMQ Message Brokers to interact together and operate as a single logical broker.

With the interbroker architecture, each broker can register with other AmbrosiaMQ Message Brokers. The broker registers as a single proxy to represent all of its clients' publications and subscriptions. The architecture supports a variety of topologies, which developers can adopt depending on their application requirements. For example, basic designs include collectives (in which each broker is connected to all the others) and hierarchies (in which individual brokers or collectives form a "tree").

This represents a fundamental expansion in terms of scalability for AmbrosiaMQ systems. It provides several overall benefits:

- scalability to support many concurrent users,
- fault isolation,
- integration of geographically dispersed offices.

Multiple AmbrosiaMQ brokers may be configured to work together in a network. This is called an Interbroker Network. An interbroker network is organized into one or more collectives. Each broker within a collective has connections to every other broker in the collective, and each collective within a network has connections to every other collective. These connections allow a subscriber to access subjects published on any broker within the network. Collectives may standalone or they may be interconnected, via a common broker, to form a collective hierarchy.



Interbroker Topology Configuration (interbroker.cfg)

The interbroker network configuration resides in a file named interbroker.cfg. The interbroker configuration may either be obtained from a local file or from a configuration server broker. For brokers using a local interbroker configuration file, the interbroker.cfg file must be located in the current directory of the AmbrosiaMQ broker process.

The interbroker configuration file consists of a broker block, followed by one or more collective blocks. The format is:

```
.brokers
<broker>,[ssl://]<host:port>

.collective <collective> <broker1>
<broker2> [...]
```

The following interbroker configuration file demonstrates a simple interbroker configuration that defines two standalone collectives.

```
.brokers
broker1, trout.acme.com:8001
broker2, bass.acme.com:8001
broker3, ssl://shark.acme.com:8002
broker4, ssl://tuna:acme.com:8002

.collective collective1
broker1
broker2

.collective collective2
broker3
broker4
```

The following interbroker configuration file demonstrates an interbroker configuration that defines two interconnected collectives.

```
.brokers
broker1, trout.acme.com:8001
broker2, bass.acme.com:8001
broker3, ssl://shark.acme.com:8002
broker4, ssl://tuna.acme.com:8002
broker5, ssl://pike.acme.com:8002

.collective collective1
broker1
broker2
broker3

.collective collective2
broker3
broker4
broker5
```

Interbroker Configuration Properties (ambroker.ini)

Property Name	Default Value	Description
ENABLE_INTERBROKER	False	Determines whether this broker will attempt to connect to other brokers in an interbroker network.
IB_CONFIG_SERVER	None	The host:port connection specification for the broker designated to act as the interbroker configuration server.

Interbroker Admin Console

The Interbroker Admin Console is a console application that allows an Administrator to dynamically create or modify interbroker networks, while the brokers are running. The Interbroker Admin Console is started with the following command (this assumes you have sourced the AmbrosiaMQ environment).

```
java com.u1.tools.IBAdminConsole <configuration broker host>:<port>
```

Once the Interbroker Admin Console has started, the tool will display the following prompt:

```
IBAdmin>
```

Pressing the *Enter* key will cause the Interbroker Admin Console to display the following menu of commands:

```
C:\AmbrosiaMQ> java com.u1.tools.IBAdminConsole localhost:8001
IBAdmin>
Commands:
    list brokers
    list collectives
    list <collective>
    new collective <collective>
    new broker <broker> <host:port>
    add <collective> <broker>
    remove <collective> <broker>
    remove collective <collective>
    remove broker <broker>
```

To use a command simply type in the command name and require parameters. The command will be executed when you press the *Enter* key. Updates to the interbroker configuration will occur as soon as the command is executed.

4

System Messages

Broker Published System Messages

Expired Message Notification

If the flag *ENABLE_MSG_EXPIRATION_NOTIFICATION* is set to true, then the broker will publish expired message subjects to *$SYS.broker.expiredMessage.<subject>.<expire time>*.

Message Subject:

$SYS.broker.expiredMessage.<subject>.<expire time>

Message Body:

Body copied from the expired message

Connect Message

Upon a client's successful connection to a broker, a message containing the broker name, remote IP address, and remote port is published to *$SYS.broker.connect.<broker name>.<uid>.<appid>*.

Message Subject:

$SYS.broker.connect.<broker name>.<uid>.<appid>

Message Body:

UTF String: broker name

UTF String: remote IP address

int: remote port

Disconnect Message

When a client disconnects normally, a message containing the remote IP address and remote port is sent to *$SYS.broker.disconnect.<broker name>.<uid>.<appid>*.

Message Subject:

$SYS.broker.disconnect.<broker name>.<uid>.<appid>

Message Body:

UTF String: remote IP address

int: remote port

Drop Message

When a client connection is abnormally terminated, a message containing the remote IP address and remote port is sent to *$SYS.broker.drop.<broker name>.<uid>.<appid>*.

Message Subject:

$SYS.broker.disconnect.<broker name>.<uid>.<appid>

Message Body:

UTF String: remote IP address

int: remote port

Reject Message

When a *connection* is rejected due to an authentication failure, messages are published to *$SYS.broker.reject.<broker name>*. If *uid* and *appid* are known, then messages are published to *$SYS.broker.reject.<broker name>.<uid>.<appid>*. The message body will contain the remote IP address, remote port, and the reject error code.

Message Subject:

$SYS.broker.reject.<broker name>.<uid>.<appid>

Message Body:

UTF String: remote IP address

int: remote port #

int: error code

Client System Messages

Connection Dropped

When a *connection* is lost or dropped without a normal disconnect, the AmbrosiaMQ client will post a system message to itself that the broker connection has been lost. Applications can use this message to implement recovery and reconnection logic.

Message Subject:

$SYS.client.brokerConnectionDropped

Message Body:

int: Error code

int: Specific error code information

A

Glossary

This glossary offers a brief overview of commonly used terms to help clarify AmbrosiaMQ's major concepts and characteristics.

Definition of Terms

AmbrosiaMQ Client Application

The client application refers to an application that uses the AmbrosiaMQ client package to communicate with the event management system. The package may already reside, pre-provisioned, on a user's local computer or may arrive with an applet downloaded from an HTTP server. Once on the local machine, an application uses methods in the client API to connect with the message broker and join the AmbrosiaMQ system.

Credentials

This consists of the user's ID (i.e., name or login) and password. Credentials objects identify the end users of AmbrosiaMQ client applications.

Event

A business event can be anything that happens which materially affects your organization. AmbrosiaMQ encapsulates business events as information carried in messages.

Event Management System

The AmbrosiaMQ Event Management System includes the entire communications infrastructure and the information flowing within the system. The infrastructure links several main components: the business application using the AmbrosiaMQ Client API, the AmbrosiaMQ Message Broker, and other client applications.

Interbroker

An optional AmbrosiaMQ feature that allows you to have more than one broker. Using the Interbroker Configuration tool, you may establish and maintain multiple brokers. Groups of brokers are then organized into "collectives."

Message

Message objects carry event information between AmbrosiaMQ client applications. AmbrosiaMQ messages consist of a subject name and a content body, which is simply a series of bytes. As a result, messages can include any application data. Each message must be published to a specific subject. AmbrosiaMQ does not set any limits on the size of a message. AmbrosiaMQ handles the details of marshalling, unmarshalling, and other low level process details so that developers can focus on the message contents.

Message Broker

As the heart of the system, the broker routes messages and implements services. A broker communicates with AmbrosiaMQ client applications; in the future, it will connect with other message brokers as well. The message broker maintains lists of connected clients and their current subscriptions. Client applications and the message broker communicate by use of the com.u1.client package.

Message Handler

Message handlers act upon messages arriving at a client. The client application must initially set a default message handler; it can then take this further by designating specific message handlers for individual subjects or sets of subjects.

Publish

A publish occurs when an AmbrosiaMQ client application produces information and sends a message with this information to the message broker.

Publish/Subscribe

The publish/subscribe communication model is based on the ability of client applications to publish messages tagged with subject names, which are then delivered only to other clients who hold a subscription (that is, an interest) for that subject. This basic model promotes a many-to-many mapping of publishers to subscribers. It also describes an anonymous communications architecture wherein subscribers do not need to know who published a particular message and publishers do not know the subscribers' specific addresses. A central message broker tracks subscriptions, routes messages, and implements delivery policies. By default, every client application can publish and/or subscribe to a subject.

Quality of Protection

The Quality of Protection (QOP) options, for a subject, consists of privacy and integrity. Privacy assures that only the intended recipient views a message by using encryption. Integrity uses a cryptographic checksum algorithm to ensure that the information within a message cannot be altered without the recipient knowing about it.

Quality of Service

Quality of Service (QOS) refers to delivery semantics, which are the levels of assurance with which AmbrosiaMQ will deliver a message. AmbrosiaMQ supports two types of delivery semantics: *reliable* and *guaranteed* message delivery.

Request/Reply

The request/reply model provides a very useful approach to query for particular information. AmbrosiaMQ implements request/reply as a special case of publish/ subscribe. When a publisher sends out a message as a synchronous request, the function will wait until the first response to the request is received; all other responses are ignored.

Session

Every session represents a single context of communication between the broker and client application. Client applications can create multiple sessions in which to perform work.

Subjects

Subjects provide the key to the routing of messages between publishers and subscribers. Subjects provide an anonymous alternative to citing specific destination addresses. Almost any string of Unicode characters can act as a subject to describe the topic category of a message. Subject names consist of one or more levels separated by the period (.) character. AmbrosiaMQ reserves subject names with "$SYS" and "$ISYS" at the first level for internal system use.

Subject Expression

A subject expression can include multiple levels and one or more wildcards, the asterisk (*) or the pound(#). Thus, the subject expression can represent a set of subjects for subscribing or binding. Wildcard characters are not interpreted as wildcards in publish(), unsubscribe(), or unbind().

Subject Tree

Subject trees form the basis of message routing in AmbrosiaMQ and thus play an important role in application design. Subject trees are hierarchical strings composed of levels of subject names. Levels are established by using a separator character, the period (.), to partition the subject tree branches. Thus, levels can establish subject branches such as "SOFTWARE.JAVA" or "SOFTWARE.C" to provide groups and specific sub-topics for messages. The system can have an unlimited number of subject trees, and each subject tree can have an unlimited number of levels.

Subscribe

To subscribe, a client application registers interest in a subject or multiple subjects, possibly by using wildcards. Subscribing presents several choices: quality of service, specific message handler, and related features. The developer usually hides the use of message handlers, delivery semantics, and specific subjects from the end user of the application using AmbrosiaMQ for its communications.

Transaction

Several classes of applications require grouping individual tasks into a single, coordinated "unit of work". This grouping of tasks is called a transaction. In a *distributed* transaction, the tasks in the unit of work are performed on physically different computers or by different application programs or services.

Two-Phase Commit

This protocol enables AmbrosiaMQ to commit a transaction and send an entire set of messages in a unit of work, or to abort and roll back the entire set of messages. Thus, the message broker acts as a resource manager so client applications can participate in transactions.