# General Architecture of EntireX Broker

This chapter covers the following topics:

- Introduction to EntireX Broker Architecture
- EntireX Broker Communication Models
- Architecture of Broker Stub
- Architecture of Broker Kernel

---

## Introduction to EntireX Broker Architecture


This section describes the command process flows within the Broker kernel and stubs when two application components communicate with each other using EntireX Broker. The Broker consists of the following components:

- a stub (application binding), which resides within the process space of each application component;
- a Broker kernel, which resides in a separate process space, managing all the communication between application components.

The details of the transport protocols remain transparent to the application components because they reside within EntireX Broker (stubs and kernel). The EntireX Broker kernel and the location of the transport protocols are the architectural aspects of EntireX Broker that distinguish it from other messaging middleware.

## EntireX Broker Communication Models

The EntireX Broker uses two communication models: client and server and publish and subscribe. Client and server communication is used if data is to be sent to exactly one partner. "Publish and subscribe" communication is used if data is to be published.

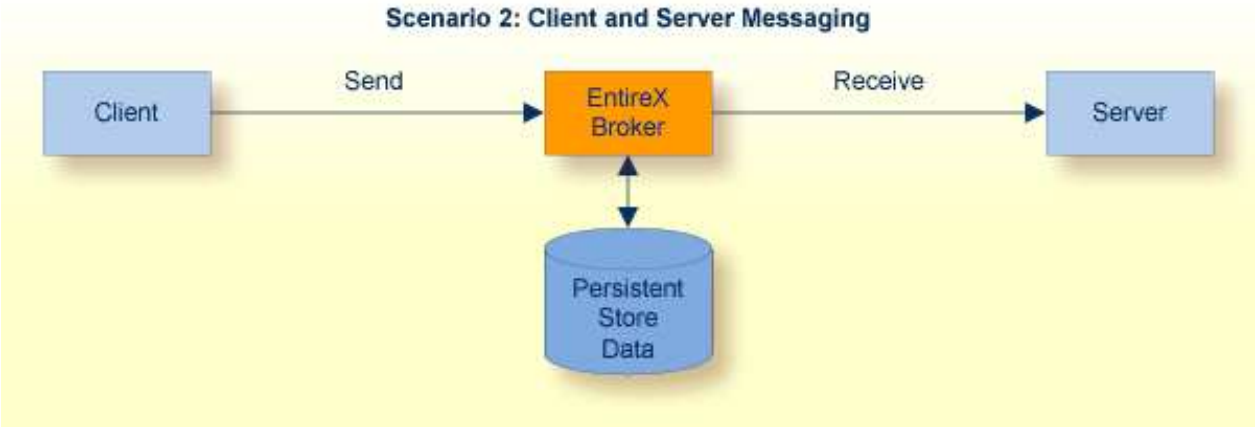
### Client and Server

See *Writing Applications: Client and Server* for details of the client and server model.

### Example Scenario 1: Client and Server Messaging (Synchronous)


This is a synchronous messaging scenario: send request and wait for a response.

### Example Scenario 2: Client and Server Messaging (Asynchronous)




This is an asynchronous messaging scenario: put message in service queue.

**Note:**
Client and server have specific meanings within the context of EntireX.

| Term | Description |
|---|---|
| Client | An application component intending to access a service makes its request via EntireX Broker which routes the request to the specific application component offering this service.<br><br>The request can be a single pair of messages comprising request/reply; or it can be a sequence of multiple, related messages containing one or more requests and one or more replies, known as a conversation. This enables EntireX Broker to be used for applications supporting different programming interfaces. It also allows interoperability between types of application components employing these different interfaces. |
| Server | An application component offering a service registers it with EntireX Broker. EntireX Broker makes the registered service available to other application components capable of communicating with EntireX Broker. The fact that a server has been registered and is available in this way defines it as a service in terms of class/name/server within the context of EntireX. |

# Publish and Subscribe

See section *Writing Applications: Publish and Subscribe* for details of the publish-and-subscribe model.

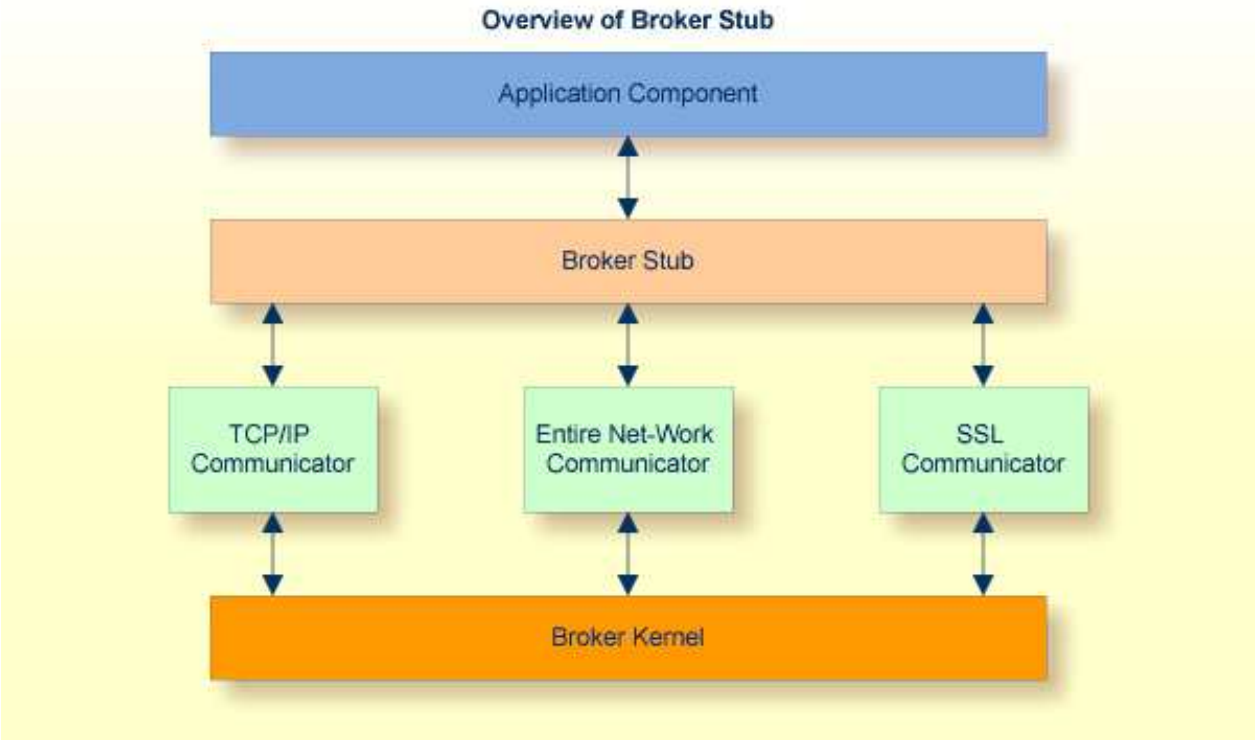

| Term | Description |
|---|---|
| Publisher | An application component acting as a publisher is able to send messages to a specified topic: These messages constitute publications which are now available to the various different subscribers. These messages are automatically kept in the persistent store if there are any subscribers with "durable" status.<br><br>Publications are retained for a specified time limit of days, months or years until all the subscribers have had the opportunity to receive them. After this time, or upon delivery to every existing subscriber, the publications are removed from the system. |
| Subscriber | An application component which is interested in one or more specific topics notifies Broker kernel, using the Subscribe command. This informs Broker that any publications sent to the specified topics will be required by this subscriber and so should be retained and then forwarded to this subscriber when this application component solicits these subscriptions. Subsequently the subscriber can issue receive commands to solicit any outstanding subscriptions.<br><br>The subscriber can subscribe to EntireX Broker with the `ALLOW-DURABLE` option which means the subscriptions are kept in the persistent store even after the Broker kernel or the application component has been restarted. |

# Architecture of Broker Stub

The type of communication model described in this section and in the section *Architecture of Broker Kernel* is client and server.

## Overview of Broker Stub

The EntireX Broker stub is another name for Software AG's ACI (Advanced Communication Interface). The stub implements an API (application programming interface) that allows programs written in various languages to access EntireX Broker.




See also *Administration of Broker Stubs* under z/OS | UNIX | Windows | BS2000/OSD | z/VSE | IBM i | OpenVMS | z/VM.

## Description of Command Process Flow within Broker Stub

The following table gives a step-by-step description of a typical command process flow from and to a Broker stub. This example describes a `SEND/RECEIVE` command pair.

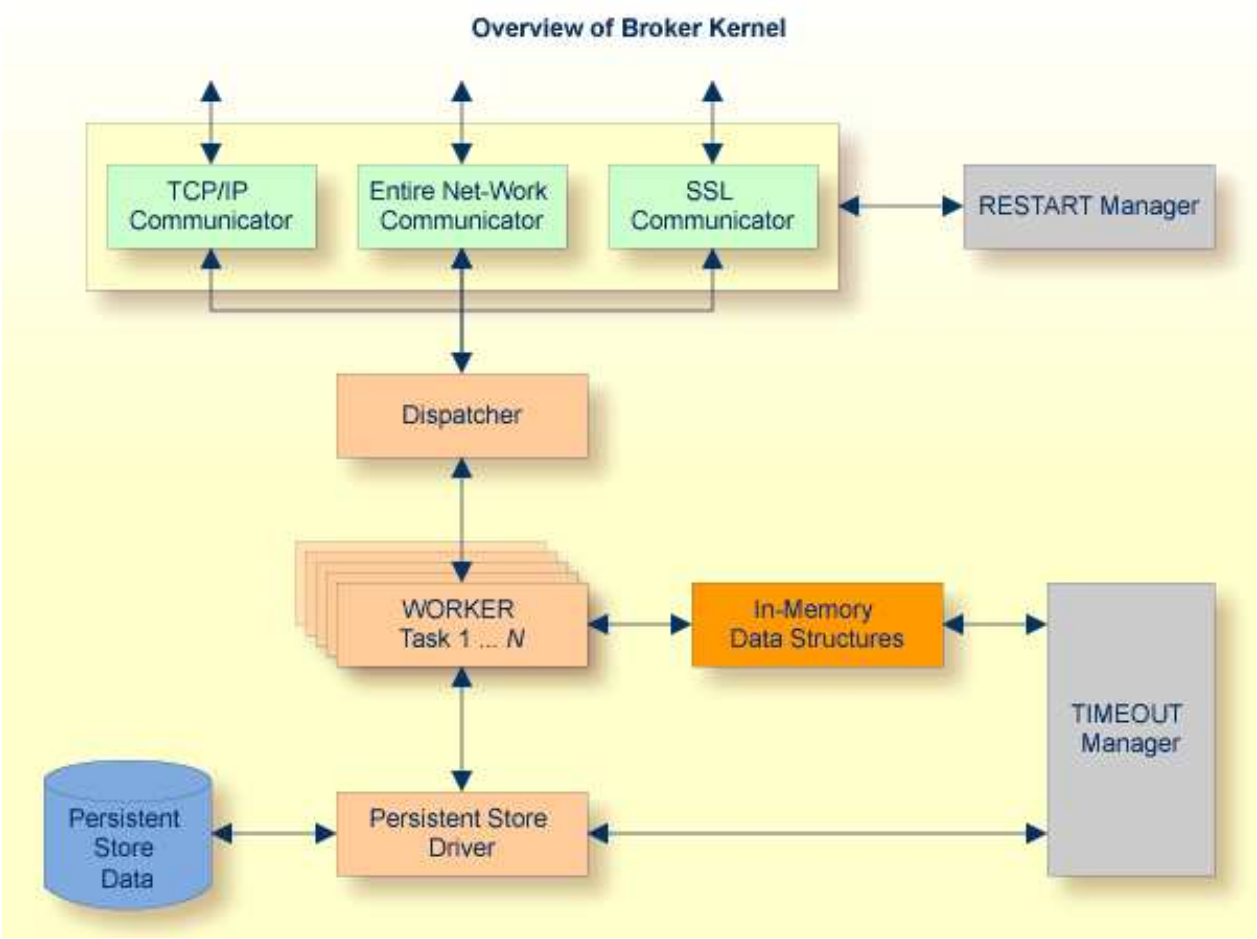
**Note:**
Publish and subscribe uses `SEND_PUBLICATION` instead of `SEND`, and `RECEIVE_PUBLICATION` instead of `RECEIVE`.

| Step | Description |
|---|---|
| 1 | The originating application program calls the stub with a `SEND/WAIT=YES` command. The stub builds the necessary information structures and communicates the message to the Broker kernel. Basic validation is performed in the stub before the command is passed to the Broker kernel. |
| 2 | The stub uses one of the following transport mechanisms to transmit the command to the Broker kernel: TCP, SSL or Entire Net-Work. The application does not have to recognize the details of the transport protocol since all transport protocol processing resides entirely within the stub. |
| 3 | The application is suspended while the stub waits for a response. Since the application has issued `SEND`, `WAIT=YES` it must wait for the message to travel via the Broker kernel to the partner application which will satisfy the request. |
| 4 | After the request has been satisfied and the message returns from the partner application, via the Broker kernel, the stub will pass control back to the originating application. |

# Architecture of Broker Kernel

The type of communication model described in this section and in the section *Architecture of Broker Stub* is client and server.

## Overview of Broker Kernel


Overview of Broker Kernel
TCP/IP Communicator
Entire Net-Work Communicator
SSL Communicator
RESTART Manager
Dispatcher
WORKER Task 1 ... N
In-Memory Data Structures
TIMEOUT Manager
Persistent Store Data
Persistent Store Driver

## Description of Command Process Flow within Broker Kernel

The following table gives a step-by-step description of a typical command process flow within the Broker kernel. This example describes a `SEND/RECEIVE` command pair.

**Note:**
Publish and subscribe uses `SEND_PUBLICATION` instead of `SEND`, and `RECEIVE_PUBLICATION` instead of `RECEIVE`.

| Step | Description |
|---|---|
| 1 | The originating application program calls the Broker stub with a `SEND` command. The stub builds the necessary information structures and transmits the message to the Broker kernel using TCP, SSL or Entire Net-Work. |
| 2 | The message is received by one of the communications subtasks running within the Broker kernel. The communications subtask passes the message to the dispatcher. |
| 3 | The dispatcher schedules the processing of the message within a worker task inside the Broker kernel. |
| 4 | Worker task processes the inbound message, performing any necessary data conversion and security operations, and then determines the partner to which the message is to be routed. Any necessary persistence operations are performed under control of the worker task. |
| 5 | The outbound message is passed to the relevant communications subtasks within the Broker kernel for transmission to the partner application component. |
| 6 | The partner application component which has issued a `RECEIVE` command via the broker stub obtains the message from the originating application program. |
| 7 | The partner application component then processes the message and normally makes a reply. |

**Notes:**

1. Application components can exchange successive related message pairs. This action constitutes a conversation.
2. Clean-up processing of timed-out commands is performed asynchronously by the Broker kernel Timeout Manager which acts upon in-memory data structures as well as data within the persistent store.
3. The communications restart manager is able to restart any communications subtasks which may have become temporarily disabled, for example by restarting the machine’s TCP/IP driver.