Writing Applications: Units of Work

This chapter describes the concept of units-of-work programming for EntireX Broker. Units of work are the precondition for achieving persistent messaging within your applications. Units of work can also be used without persistence.

This chapter assumes you are familiar with basic Broker ACI programming. If you are not familiar with it, we recommend beginning with the chapter *Writing Applications: Client and Server*.

This chapter covers the following topics:

- What is a Unit of Work?
- Control Block Fields and Verbs
- Client/Server Programming for Units of Work
- Client/Server Programming for a Persistent Unit of Work
- Client/Server Restart after System Failure

What is a Unit of Work?

A unit of work (UOW) is a group of related messages transmitted and received as a single entity. This is achieved through the sender committing as a single unit all the messages being sent and the receiver acknowledging receipt, as a single unit, of all the messages being received. Units of work are used in conjunction with conversations where a UOW exists strictly within one conversation. There can be more than one unit of work within a conversation. Where this is the case, subsequent UOWs can be created by either the client or the server. Since the conversation is always initiated by a client, the first UOW in the conversation is always created by the client. The UOW creator must commit the UOW to be created before being allowed to create another UOW within the same conversation.

Messages belonging to a UOW are always sent with `OPTION=SYNC`, or `OPTION=COMMIT`, which performs an implicit `COMMIT` at the same time as the `SEND`. Messages belonging to a UOW are always sent asynchronously, i.e. `SEND`,`WAIT=NO`. Messages belonging to a UOW are always received with `OPTION=SYNC` and can be received either with `WAIT=NO` or by specifying `WAIT`=[`YES` | *`timevalue`*], depending on application requirements.

Control Block Fields and Verbs

- Basic Functionality of Broker API
- ACI Syntax
- Key ACI Field Names
- Key Verbs for FUNCTION Field

Basic Functionality of Broker API

This section describes the expanded functionality of the Broker API used when programming units of work (UOWs) with or without persistence.

- **DEREGISTER**
  The function `DEREGISTER` is used by a server to indicate its intention to terminate its role as a server for the specified `SERVER-CLASS`, `SERVER-NAME` and `SERVICE`. The server can terminate its role as server for all class, server and service names for which it is registered, using a single `DEREGISTER`.

- **RECEIVE**
  The function `RECEIVE` is used by the server to obtain new requests from a client, and in the case of conversations, to obtain subsequent related messages from the same client. This function is also used by clients that issue asynchronous requests and wish to obtain the server's reply at a later time. The field `CONV-ID` defines the behavior of this function. `RECEIVE,CONV-ID=NEW` signals the server's readiness to obtain the next available new request, whereas the value `CONV-ID`=*nnn* indicates that the next message within an existing conversation is being requested by the server. The client uses `RECEIVE,CONV-ID`=*nnn* to obtain asynchronously a reply from the server for an existing conversation.

- **REGISTER**
  The function `REGISTER` is used by a component of an application to identify its intention to become a server and satisfy requests issued to the named `SERVER-CLASS`, `SERVER-NAME SERVICE`.

- **SEND**
  The function `SEND` is used by the client either to initiate a new conversation or to send subsequent messages within that conversation. This function is also used by servers to reply to the client during the course of a conversation. Each message is assigned to the unit of work currently being created by the sender. If this is the first message from the sender, a new UOW is created. Senders can create a subsequent unit of work by committing their existing UOW, creating and performing another subsequent `SEND` function. The field `CONV-ID` defines the behavior of this function regarding conversations. The client uses `SEND,CONV-ID=NEW` to initiate a new conversation and the value `CONV-ID`=*nnn* when sending subsequent related messages in a conversation. The server always uses `SEND,CONV-ID`=*nnn* when replying to a client, where *nnn* indicates the identity of the existing conversation. The `SEND` command is always used asynchronously with units of work, by both client and server. The sender can override the default persistence setting in the attribute file for the server class, server name and service, using the ACI field `STORE`.

- **SYNCPOINT**
  The function is used by either the client or the server when committing UOWs that they are creating, and also to acknowledge receipt of UOWs that they are receiving. It can also be used by the creator

of a UOW to determine its current status or modify the status of a UOW at a later time.

ACI Syntax

Function	Fields in EntireX Broker Control Block
DEREGISTER	API = 1 or higher , BROKER-ID = broker_id , USER-ID = user_id [,TOKEN = token] , SERVER-CLASS = class_name | * , SERVER-NAME = server_name | * , SERVICE = service_name | * [,OPTION = QUIESCE | IMMED]
RECEIVE	API = 3 or higher for UOW , BROKER-ID = broker_id , USER-ID = user_id [,TOKEN = token] , OPTION = SYNC , WAIT = n | YES | NO , CONV-ID = conv_id | NEW | OLD | ANY , SERVER-CLASS = class_name | * , SERVER-NAME = server_name | * , SERVICE = service_name | * [,USTATUS = user_status] [,UOWID = uowid]
REGISTER	API = 1 or higher , BROKER-ID = broker_id , USER-ID = user_id [,TOKEN = token] , SERVER-CLASS = class_name, , SERVER-NAME = server_name, , SERVICE = service_name
SEND	API = 3 or higher for UOW , BROKER-ID = broker_id , USER-ID = user_id [,TOKEN = token] , OPTION = COMMIT | SYNC , WAIT = NO , CONV-ID = conv_id | NEW , SERVER-CLASS = class_name, , SERVER-NAME = server_name, , SERVICE = service_name [,USTATUS = user_status] [,STORE = BROKER | OFF] [,UWTIME = uow_life_time] [,UWSTATUS-PERSIST = uow_status_persist_multiplier | UWSTAT-LIFETIME = uow_status_persist_lifetime] [,UOWID = uowid]

Function	Fields in EntireX Broker Control Block
SYNCPOINT	API = 3 or higher for UOW , BROKER-ID = broker_id , USER-ID = user_id [,TOKEN = token] , OPTION = BACKOUT | CANCEL | COMMIT | DELETE | EOCCANCEL | LAST | QUERY | SETUSTATUS [,CONV-ID = conv_id] [,UOWID = uowid] [,USTATUS = user_status]

Key ACI Field Names

ACI Field Name	Explanation
SERVER-CLASS	A client uses these fields to identify the service that it requires. A server uses this to offer a service.
CONV-ID	Identifier to obtain and specify the conversation. Also used to determine communication mode (non-conversational or conversational).
FUNCTION	Function code for one of the verbs; see *Key Verbs for FUNCTION Field*.
OPTION	Indication of specific Broker behavior, depending on the function.
UOWID	Identifier generated by the Broker that identifies to the caller the unit of work ID. Specify valid UOWID to indicate an existing unit of work or leave blank when starting to SEND or RECEIVE a new unit of work. It is optionally specified when examining the status of a unit of work already created by the participant.
WAIT	Time value to specify blocking or non-blocking of the conversation. See *Blocked and Non-blocked Broker Calls* for client and server | publish and subscribe.

Key Verbs for FUNCTION Field

Verb	Description
REGISTER	Inform the broker that a service is available.
RECEIVE	Retrieve request from partner.
SEND	Send reply to the partner.
DEREGISTER	Remove the availability of the service.

Client/Server Programming for Units of Work

The figure below illustrates the logical program flow of a simple two-message client request UOW and a one-message server reply UOW. See also *Broker UOW Status Transition*.

1. The server logs on, registers, and issues a `RECEIVE` operation, and waits for a new CID and a UOW (unit of work).
2. The client logs on, creates a new UOW and a new conversation ID. It sends a message as part of a UOW and then commits the UOW, allowing the Broker to deliver it.
3. The server receives the first message in the UOW. Then the next (last) message. The server then creates a new UOW for the reply. The new UOW is part of the existing conversation (CID=123). The server commits both UOWs, i.e., the incoming UOW is processed and the outgoing UOW is ACCEPTED.
4. The client receives the incoming message and commits the UOW. The UOW is now PROCESSED.

Client	**Server**

	`LOGON,UID=,TOKEN=`
	`>OK`
	`REGISTER`
	`>OK`
	`RECEIVE,CID=NEW,OPT=SYNC,WAIT=1M`
	This receive operation will be satisfied by a new CID and a UOW. Non-UOW messages will not satisfy. (waits)

`LOGON,UID=,TOKEN=`
`>OK`
`SEND,OPT=SYNC,CID=NEW,WAIT=NO`
Creates a new UOW and a new CID.	

`>OK,CID=123,UOWSTATUS=RECEIVED,`
`UOWID=987`
`SEND,OPT=SYNC,CID=123,WAIT=NO`
Adds another message to the open UOW	

`>OK,CID=123,UOWSTATUS=RECEIVED,`
`UOWID=987`
`SYNCPOINT,OPT=COMMIT,CID=123`
Commits the open UOW, allowing the broker to deliver it.	

`>OK,CID=123,UOWSTATUS=ACCEPTED,`
`UOWID=987`
UOW (UOWID=987) is now safely in the hands of the broker.
	`>OK,CID=123,UOWSTATUS=FIRST,UOWID=987`
	The initial receive operation is completed, indicating a CID, a UOWID, and the FIRST message of a UOW.

`RECEIVE,CID=123,OPT=SYNC,WAIT=1M` This will be satisfied by a UOW on CID=123.	`RECEIVE ,CID=123,OPT=SYNC` Request the next message in open UOW.

Client	Server
(waits)	`>OK,CID=123,UOWSTATUS=LAST,UOWID=987`
	Receive the next message, which is the last. The server now has all the data.
	`SEND,OPT=SYNC,CID=123,WAIT=NO`
	Create a new UOW for the reply, on CID=123.
	`>OK,CID=123,UOWSTATUS=RECEIVED,UOWID=456`
	There are now actually 2 open UOWs (987 and 456), one in each direction.
	`SYNCPOINT,OPT=COMMIT,CID=123, UOWID=`
	This commits both UOWs, the incoming one (987) is now PROCESSED and the outgoing one (456) is ACCEPTED.
`>OK,CID=123,UOWSTATUS=ONLY,UOWID=456`	`>OK,CID=123,UOWSTATUS=ACCEPTED, UOWID=456`
Receive a message, the only one, in a UOW on CID=123. This is a different UOW than was sent.	
`SYNCPOINT,OPTION=COMMIT,CID=123`	(Loops back and reissues original receive)
This commits the UOW; it is now PROCESSED	
`>OK,CID=123,UOWSTATUS=PROCESSED, UOWID=456` `LOGOFF` `>OK`	

Client/Server Programming for a Persistent Unit of Work

The figure below illustrates the logical program flow of a simple one-message persistent UOW with deferred delivery to a server, with no reply. The client queries the status of the UOW to determine its completion. See also *Broker UOW Status Transition*.

1. The client logs on and creates a new persistent UOW and a new conversation. The intended server is not currently available.
2. The client commits the open UOW, allowing the Broker to deliver it. The UOW (UOWID=987) is now stored by the Broker. It will be delivered whenever the server is available and will be retained even in case of system failure (that is, the UOW is persistent).
3. The client logs off.
4. The server logs on and registers. It receives the new conversation ID and the new UOW. The UOW is committed. Its status is now PROCESSED.

5. The client logs on using a user ID and token to identify itself as the client that originated the UOW. It then queries the status of its UOW. The status PROCESSED is returned, so the client knows that its UOW has been successfully delivered and processed by the server.

Client

```
LOGON, UID=,TOKEN=
>OK
SEND,OPT=SYNC,CID=NEW,WAIT=NO,
STORE=BROKER,
UWTIME=5M,UWSTATP=5
```

Creates a new persistent UOW and a new CID. The UOW will have a lifetime of 5 minutes; the duration of the status is 5 times this value (25 minutes). The intended server is not up at this time.

```
>OK,CID=123,UOWSTATUS=RECEIVED,UOWID=987
SYNCPOINT,OPT=COMMIT,CID=123
```

Commit the open UOW, allowing the broker to deliver it.

```
>OK,CID=123,UOWSTATUS=ACCEPTED,UOWID=987
```

UOW (UOWID=987) is now safely in the hands of the broker. The UOW will be delivered whenever the server comes up, even if the system should fail.

```
LOGOFF
```

The client can now terminate, knowing that the UOW will be delivered.

Server

Some time later, the server comes up.

```
LOGON,UID=,TOKEN=
>OK
REGISTER,
>OK
RECEIVE,CID=NEW,OPT=SYNC
```

This receive operation will be satisfied by a new CID and a UOW. Non-UOW messages will not satisfy.

```
>OK,CID=123,UOWSTATUS=ONLY,UOWID=987
```

The receive completes, indicating a CID and the ONLY message of a UOW.

```
SYNCPOINT,OPT=COMMIT,CID=123,
UOWID=987
```

This commits the UOW; its status is now PROCESSED.

```
>OK,CID=123,UOWSTATUS=PROCESSED,
UOWID=987
```

(Loop back and reissue original receive, if desired, or terminate)

Client	Server
Some time later, the client can come back and check the status of its UOW.	
`LOGON, UID=, TOKEN=`	
Specifying the same UID/TOKEN ensures that this client can be identified as the original client.	
`>OK`	
`SYNCPOINT,OPTION=LAST`	
Request the status of the last UOW this user created. The request must be made within 30 minutes, based on the value of the original SEND.	
`>OK,UOWID=987,CID=123,UOWSTATUS=PROCESSED`	
The client now knows that its UOW was successfully processed by the server.	
`LOGOFF` `>OK`	

Client/Server Restart after System Failure

Warning:
`USER` and `TOKEN` must be specified when using persistent units of work (UOWs) to persist either a message or the status of a message exchanged between partner application components, where this information is held in the persistent store.

EntireX Broker provides a reconnection feature, using the `TOKEN` field in the ACI. If the application supplies a token along with `USER-ID`, the processing is automatically transferred when a request with the same user ID and token is received, either from the same process or from a different process or thread.

You need to specify `USER` and `TOKEN` to reconnect with the correct user context after a broker has been stopped and restarted when using units of work.