Using Persistence and Units of Work

This chapter describes implementation issues and how to use persistence and units of work in EntireX Broker. It assumes you are familiar with EntireX Broker from both an administrative and an application perspective, and with the ACI programming in particular. See also *EntireX Broker* and *EntireX Broker ACI Programming*. This chapter covers the following topics:

- Implementation Issues
- Using Units of Work
- Using Persistence
- Using Persistent Status
- Recovery Processing

Implementation Issues

- Table of Persistent Store Drivers
- Changes are Required
- Attributes used for Units of Work
- ACI Fields used for Units of Work
- ACI Function SYNCPOINT used for Units of Work
- Options used for UOW Operations
- CID Implementation: Numeric Digits, Characters 0-9 and A-Z

Table of Persistent Store Drivers

A persistent store driver is an executable, or a load module that implements access to the physical persistent store. There is one persistent store driver for each persistent store type. The following table shows the persistent store options:

Persistent Store Type	Description	Operating System	Notes
Adabas	Uses Adabas database.	UNIX, Windows, z/OS, z/VSE	Adabas, Software AG's ADAptable dataBASe, is a high-performance, multithreaded, database management system.
DIV	Uses IBM Data In Virtual facility on z/OS.	z/OS	This persistent store option is implemented as a VSAM linear data set.
CTREE	c-tree© is an embedded local database that can be used as your persistent store.	UNIX and Windows	c-tree© is the fast and reliable embedded database of FairCom Corporation®.

Changes are Required

It is important to note that some level of both application and system changes are necessary to utilize UOWs. Existing message-based Broker applications will continue to operate as before.

Attributes used for Units of Work

The following table represents the keyword parameters that can be used in the Broker attribute file for UOWs. A short form of the keyword is given if applicable. Default values are underlined.

Keyword	Value	Description
`STORE`	`OFF` | `BROKER`	Broker: sets default `STORE` attribute for all units of work. Service: sets default `STORE` attribute for units of work sent to the service.
`MAX-UOWS` or `MUOW`	`0` | *n*	Broker: maximum number of active UOWs. If 0 is specified, then the Broker will not support any UOW operations. Service: maximum number of active UOWs for a service.
`MAX-MESSAGES-IN-UOW` or `UMSG`	`16` | *n*	Broker: maximum number of messages in a UOW. Service: maximum number of messages in a UOW for the service.

Keyword	Value	Description
PSTORE	NO | HOT | COLD | WARM	Broker only. Startup value for persistent store. **NO** No persistent store. **HOT** Persistent UOWs are restored to prior state during initialization. **COLD** Persistent UOWs are not restored during initialization, and the persistent store is considered empty. **WARM** (Internal Use Only) persistent UOWs are not restored during initialization, but the persistent store remains intact.
UWSTATP	0 - 254	Broker: persistent status is maintained either for persistent or non-persistent UOWs. Service: persistent status is maintained either for persistent or non-persistent UOWs for a service.
UWTIME	1D | *n*S | *n*M | *n*H | *n*D	Broker: defines the lifetime of a UOW in seconds, minutes, hours or days. This value is the time that it can remain in the system without being completed. If the UOW is not completed within this time, it is deleted with a status of TIMEOUT Service: defines the lifetime of a UOW for a service.
MAX-UOW-MESSAGE-LENGTH	*n* | 31647	Broker: defines the default maximum message size that can be sent. Service: defines the maximum message size that can be sent to a service.
DEFERRED	NO | YES	Broker: sets the default DEFERRED attribute for all services. UOWs can be sent to a deferred service even if the service is not registered. Service: sets the DEFERRED attribute for a service.

ACI Fields used for Units of Work

The following fields have been added to the broker ACI control block. Note that the actual field names may differ slightly depending on the programming language being used.

Keyword	Description
STORE	Indicates whether the specified UOW is persistent or not: **OFF** The sender accepts the persistence option specified by the service or Broker (this is the default value). **BROKER** The sender wants persistence. **NO** The sender does not want persistence, even if the service or Broker default is persistence. Also returned with RECEIVE to indicate if the UOW being received is persistent or not.
UWTIME	The amount of time that the UOW can remain incomplete without being timed out. This is also referred to as the UOW lifetime.
STATUS	The current status of a UOW. The status is returned on SEND, RECEIVE, and SYNCPOINT operations. Applicable values are as follows: **RECEIVED** One or more messages have been sent as part of a UOW but the UOW is not yet committed. **ACCEPTED** The UOW has been committed by the sender. **DELIVERED** The UOW is currently being received. **BACKEDOUT *** The UOW was backed out prior to being committed by the sender. **PROCESSED *** the receiver of the UOW has committed it. **CANCELLED *** the receiver of the UOW has cancelled it. **TIMEOUT *** the UOW was not processed within the specified time. **DISCARDED *** The UOW was not persistent and its data was discarded over a restart. * The status values marked with an asterisk are persistent, and will only exist for UOWs with persistent status. In addition, the following status values are returned on a RECEIVE operation to indicate if the message being received is part of a UOW or not, and if so, which part: **RECV_NONE** The message is not part of a UOW. **RECV_FIRST** The message is the first message in a UOW. **RECV_MIDDLE** The message is not the first or last message in a UOW. **RECV_LAST** The message is the last message in a UOW. **RECV_ONLY** The message is the only message in a UOW. All RECV_ values except RECV_NONE reflect an actual UOW status of DELIVERED.
USTATUS	A user-defined status associated with a UOW. It can be specified as part of a SEND, RECEIVE, or SYNCPOINT operation and will be returned whenever the status of a UOW is queried. See *Using User Status* below for more information.

Keyword	Description
UOWID	A unique identifier for a unit of work. This value is returned on SEND and RECEIVE operations and may be provided on SYNCPOINT operations that are querying status of UOWs.
UWSTATP	A numeric value indicating the lifetime value for persistent status. This value is a multiplier against the UWTIME value. Applicable values are: **0** Use the default specified for the service or broker. **1-254** Use 1 to 254 times the UWTIME value as the status lifetime. **255** The sender does not want persistent status, even if the service or broker default is persistent status.

ACI Function SYNCPOINT used for Units of Work

The SYNCPOINT function deals exclusively with UOWs. The following table lists the OPTION values that can be used with the SYNCPOINT function, and the associated behavior and restrictions of each one.

Note:
In many cases, the behavior will be different depending on whether the issuer is the sender or the receiver of the UOW.

Option	Caller	Behavior and Restrictions
BACKOUT	Sender	If the specified UOW is in RECEIVED status, it will be put into BACKEDOUT status. If persistent status is not specified, no trace of the UOW will remain.
	Receiver	If the specified UOW is in DELIVERED status, it will be put back into ACCEPTED status and its attempted delivery count will be incremented.
CANCEL	Sender	If the specified UOW is in ACCEPTED status, it will be put into CANCELLED status. If persistent status is not specified, no trace of the UOW will remain.
	Receiver	If the specified UOW is in DELIVERED status, it will be put into CANCELLED status. If persistent status is not specified, no trace of the UOW will remain.
COMMIT	Sender	If the specified UOW is in RECEIVED status, it will be put into ACCEPTED status. It is now available to be received by the other partner.
	Receiver	If the specified UOW is in DELIVERED status, it will be put into PROCESSED status. If persistent status is not specified, no trace of the UOW will remain.
	Both	This is a special case of the COMMIT option, where the caller specifies UOWID=BOTH in the request. This allows the caller to commit two UOWs, one being received and one being sent, in a single atomic operation.
DELETE	Sender	Deletes the persistent status of the specified UOW. The UOW must be complete and must have been created by the caller. After this request, no trace of the UOW will remain.
EOC	Sender	Commits the UOW and sets an EOC indication on the associated conversation. See COMMIT for additional information and restrictions.
EOCCANCEL	Sender	Commits the UOW and sets an EOC-CANCEL indication on the associated conversation. See COMMIT for additional information and restrictions.
LAST	Sender	Returns the status of the last UOW sent by the caller. In addition, CLASS/SERVER/SERVICE details of the associated server are also returned. The CONV-ID can be included to qualify the request.
QUERY	Sender	With UOWID=n, returns the status of the specified UOW. In addition, CLASS/SERVER/SERVICE details of the associated server are also returned.
SETUSTATUS	Both	Updates the user status field of the specified UOW. The UOW must be in RECEIVED, ACCEPTED, or DELIVERED status.

Options used for UOW Operations

This table lists option values used to support UOW operations:

Option	Function	Behavior and Restrictions
SYNC	SEND	This option indicates that the data being sent is part of a UOW. The UOW is created on the first send, and subsequent sends will add messages to the UOW.
SYNC	RECEIVE	This option indicates that the RECEIVE can be satisfied only with a message in a UOW.
MSG	RECEIVE	This option indicates that the RECEIVE can be satisfied only with non-UOW messages.
ANY	RECEIVE	This option indicates that the RECEIVE can be satisfied by either a non-UOW or a UOW message. It is up to the receiver to determine which, based on the UOWSTATUS field that is returned.
COMMIT	SEND	This option combines a SEND and a SYNCPOINT, OPTION=COMMIT into a single operation. It allows the sender to create and commit a UOW in a single operation.

CID Implementation: Numeric Digits, Characters 0-9 and A-Z

In order to support unique conversation identifiers at Broker restart, there is an implementation of the CID which is alphanumeric and an internal identifier.

Note:
The CID is a Broker-generated identifier for the conversation, and the application should not make any assumptions about either the content or format of all or any part of the CID field, or about any relationship between CIDs.

If any of the following three conditions exist, the all-numeric implementation of the CID field will be used in order to ensure compatibility:

- the Broker does not support any UOW processing;
- the application program is using API_VERSION 1 or 2 in its request;

  or

- the target service does not support UOWs.

Note:
This level of compatibility may be removed at some point in the future.

Using Units of Work

- UOW vs non-UOW Conversations
- Use of LOGON and TOKEN
- User Identification for Units of Work
- Which Applications should use UOWs?

- Understanding UOW Status
- UOW Status on RECEIVE
- Using User Status
- Resource and Performance Considerations

UOW vs non-UOW Conversations

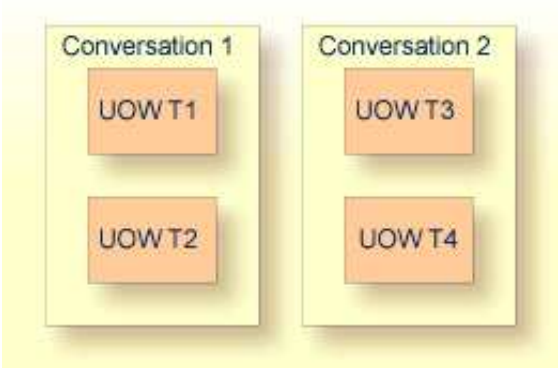
A Broker conversation will support either UOWs or messages, but not both. At the time the conversation is created, the Broker will determine which is to be supported.

Sequencing of Messages across Conversations

The order of delivery of new conversations to receivers is determined by the `COMMIT` time of the first UOW within its conversation. The conversation delivered to the receiver first is the one containing the first UOW for which the sender issues a `SEND,OPTION=COMMIT` or `SYNCPOINT,OPTION=COMMIT`.

If there is more than one UOW in a conversation, the `COMMIT` time of the first UOW determines the age of that conversation. Also, multiple UOWs within a conversation are picked up by the receiver, in the same sequence as they were committed by the sender.

Scenario: A server starts to receive UOWs (`CONVID=NEW`) and receives UOW T1 first, since this UOW is committed first. If the server issues another receive (`CONVID=NEW`), it receives UOW T3. If, however, the UOWs are not combined in conversations (i.e., every UOW is in a separate conversation), the server receives (`CONVID=NEW`) UOW T1 first, then UOW T2, UOW T3, etc.



The `COMMITTIME` field in the Broker control block shows `COMMITTIME` of the first UOW in a conversation.

Use of LOGON and TOKEN

An explicit `LOGON` function must be used before a program can use any of the UOW functions. In order to enable client and server programs to recover the status of their UOWs in the event of a failure (Broker, system, or application), these programs must specify a `TOKEN` value at the time of logon.

User Identification for Units of Work

EntireX Broker identifies participants by ACI fields `USER-ID` and `TOKEN` if `TOKEN` is supplied or by `USER-ID` and internal ID (so-called physical user ID) if `TOKEN` is not supplied. However, the implementation of persistent units of work is based on the user identification `USER-ID` and `TOKEN`.

Warning:
In order to avoid unpredictable inconsistencies, all applications using persistent units of work must use this user identification to run correctly. The ACI verification routines do not restrict usage of UOWs to `USER-ID` and `TOKEN` yet. Modify your application accordingly.

Which Applications should use UOWs?

Applications that should consider using UOWs fit into a couple of different categories.

- Applications that currently use multiple messages to communicate a single request are good candidates for UOWs. Grouping these messages within a UOW can give the application additional control over how its data is processed.
- Applications that intend to utilize deferred services, persistence, or persistent status must use UOWs, since these facilities are not available to message-based applications.

Understanding UOW Status

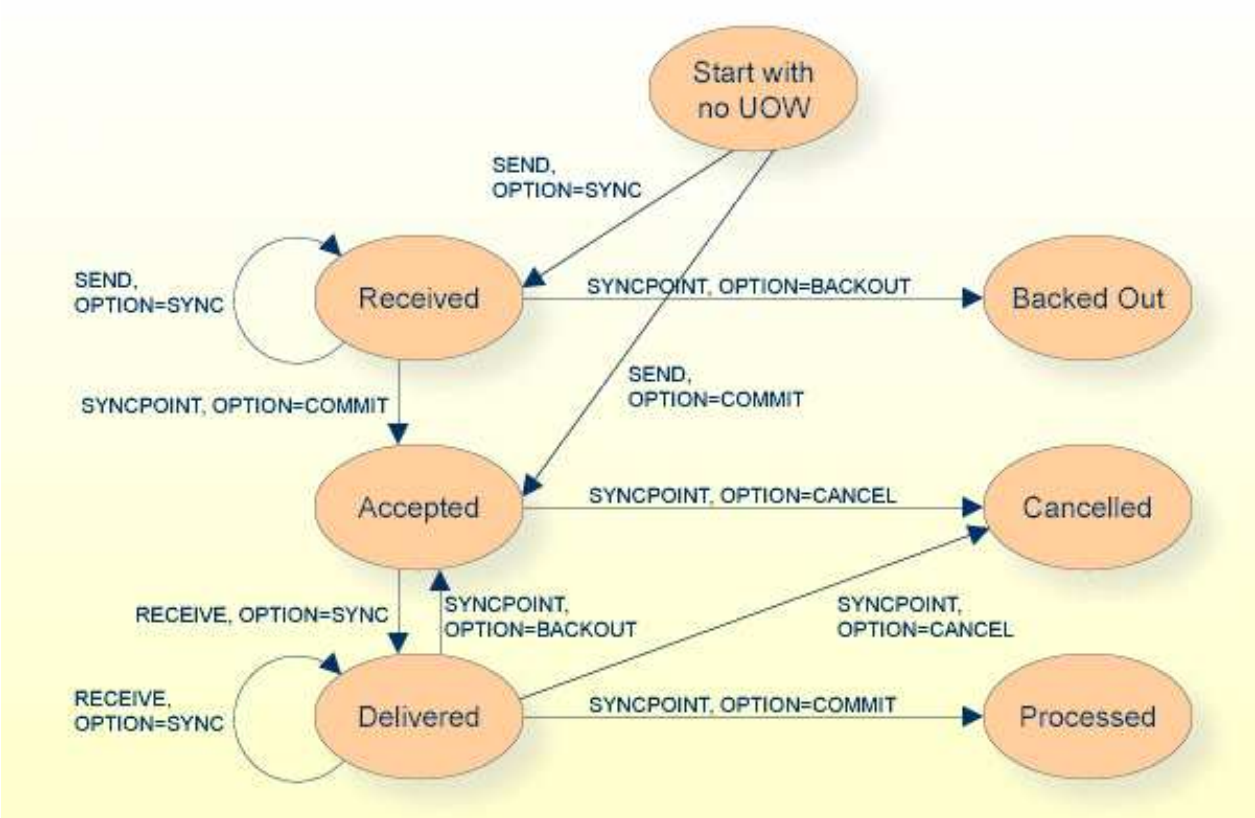
In order to use UOWs effectively, you need to understand

- the meaning of the various UOW status values;
- how they change based on events within the system;

  and

- how these changes are influenced by both persistence and persistent status.

The diagram below represents the normal status values as a UOW progresses through the system. These statuses and the transitions between them are not affected by either persistence or persistent status. The status values are indicated in ovals.



Normal Status Values as a UOW progresses through System

Note:
The UOW is available to be received when it is first committed. The status values `BACKEDOUT`, `CANCELLED` and `PROCESSED` are valid only if there is persistent status.

UOW Status on RECEIVE

When a `RECEIVE` is issued for a message within a UOW, you might expect that the UOW status returned would be `DELIVERED`, since this is the actual status of the UOW. This is not the case, however. On a `RECEIVE`, the Broker returns a special UOW status that reflects additional information about the message and the UOW. These statuses are:

- `RECV_FIRST=` the message is the first message in a UOW.
- `RECV_MIDDLE=` the message is not the first or last message in a UOW.
- `RECV_LAST=` the message is the last message in a UOW.
- `RECV_ONLY=` the message is the only message in a UOW.
- `RECV_NONE=` the message is not part of a UOW. This status is particularly useful if the application is receiving both messages and UOWs.

If you receive a status of either `RECV_LAST` or `RECV_ONLY` and then issue another `RECEIVE` for the same UOW, you will get an error `00740301 Conversation found: end of unit of work` indicating the end of the UOW.

Using User Status

The user status field of the UOW allows additional, application-specific information to be carried with the UOW. It can be used to maintain status or indicate error information. It can also provide a form of "out-of-band" data communication between the sender and the receiver of a UOW.

For example, if a server is processing a long-running UOW, it can periodically update the user status of the UOW (using `SYNCPOINT, OPTION=SETUSTATUS`) to indicate its progress. The client can periodically get the user status (using `SYNCPOINT, OPTION=QUERY`) and report the progress back to the end-user.

As another example, the sender of a long-running UOW can update the user status to indicate that processing of the UOW should be abandoned by the server. The server can periodically get the user status while processing and react accordingly.

Resource and Performance Considerations

Each active UOW consumes memory resources (approximately 140 bytes per UOW) in a preallocated pool, not including the size of the message itself.

Also, additional memory resources such as the conversation and participant control blocks for the UOW, together with messages associated with them, will remain in memory for a deferred service when persistence is used. This can become significant when UOWs are being sent to a deferred service. However, the message itself does not remain in memory if sent to a service which is not currently registered - the whole purpose of deferred services. If the service is currently registered, the message remains in memory.

Messages that are sent to any (registered or unregistered) service can be "paged out" by Broker if storage is required. This feature considerably eases memory consumption when using persistence.

Using Persistence

- When do Persistent UOWs make Sense?
- Adding Persistence to a UOW
- Resource and Performance Considerations
- Which Information is saved with the UOW?
- What happens when Broker restarts?
- UOWs and Replicated Servers

When do Persistent UOWs make Sense?

A UOW should be made persistent if the sender wants the Broker to assure that the UOW will be deliverable, even if there is a system or Broker failure. Assured delivery assumes that the intended receiver of the UOW is active, or becomes active within the specified lifetime of the UOW.

Since most existing Broker applications are interactive, they are probably not good candidates for persistent UOWs. New application models can now be implemented, using persistent UOWs. For example, a service that collects information from other services, such as accounting, inventory, logging, etc., would be a good fit for persistent UOWs. Another example could be a client sending a long-running request to a service (one that may be inactive or busy), disconnecting, and coming back some time later to retrieve the results. The reliability of assured delivery makes this model practical.

Persistent UOWs do not require persistent status.

Adding Persistence to a UOW

A UOW can be made persistent:

- by specifying `STORE=BROKER` in the ACI request that creates the UOW;
- by specifying `STORE=BROKER` in service definition or service defaults portion of the Broker attribute file, making all UOWs for that service persistent; or
- by specifying `STORE=BROKER` in the Broker defaults section of the Broker attribute files, making all UOWs in the system persistent.

In addition, specifying `STORE=NO` in the ACI request that creates the UOW will explicitly make the UOW non-persistent, overriding any Broker or service default.

Resource and Performance Considerations

A persistent UOW consumes resources in two areas.

- When the UOW is committed by the sender, all of the messages are written to the persistent store. This will generate multiple I/O operations, depending on the number and size of the messages.
- Space used to store the UOW and its messages will be allocated in the persistent store and will remain until the UOW is completed.

Performance of certain specific functions (e.g. `SYNCPOINT OPTION=COMMIT` by the sender of a UOW) will be affected by the additional time required to perform the I/O operations associated with writing the UOW and message(s) to the persistent store. These operations are performed synchronously because the Broker must ensure that the UOW, once committed, can be recovered in the event of a system or Broker failure.

Which Information is saved with the UOW?

When the UOW is initially created in the persistent store, the following information is written:

- Unit-of-work ID
- Conversation ID
- UOW Sender information, including:
    - User ID

    - Token
    - Server/service/class *
- UOW receiver information, including:
    - User ID **
    - Token **
    - Server/service/class *
- Creation timestamp
- UOW lifetime value
- Persistence and persistent status values

The following pieces of information will be included when the UOW is initially written to the persistent store and will be updated, as needed, during the life of the UOW:

- UOW status
- UOW user status
- Attempted delivery count
- Number of messages in UOW
- Total message size in UOW
- Persistent status lifetime value
- Conversation state and EOC reason code

* Server/service/class information is only saved if the sender or receiver is a registered service.

** The receiver's user ID and token are only saved if the receiver is a service that has already acquired the conversation associated with this UOW. When there are multiple instances of a service, this means that a new conversation can be restarted by any instance of the service, but an existing conversation is bound to a specific instance of the service.

What happens when Broker restarts?

- Restart Behavior of UOW
- Re-creation of Internal Control Blocks
- Behavior of Conversation at Broker Restart

Note:
"Restored" is an active UOW which has been returned to `ACCEPTED` status; "Discarded" is a UOW which has not been returned to `ACCEPTED` status. "Discarded" does not imply the status of `DISCARDED`.

Warning:
The persistent store must be available before you attempt to restart your Broker; otherwise your Broker will not restart.

Restart Behavior of UOW

- **Restart Table 1**
  The behavior during restart of the following states depends on the previous settings of the options Persistent UOW and Persistent Status.

UOW Status before Restart	Persistent UOW: YES | NO	Persistent Status: YES | NO	Behavior of UOW and Status	UOW Status after Restart *
RECEIVED	YES	YES	UOW not restored; Status is restored	BACKEDOUT
RECEIVED	YES	NO	UOW not restored; Status not restored	---
RECEIVED	NO	YES	UOW not restored; Status is restored	DISCARDED
RECEIVED	NO	NO	UOW not restored; Status not restored	---
ACCEPTED	YES	YES	UOW is restored; Status is restored	ACCEPTED
ACCEPTED	YES	NO	UOW is restored; Status is restored	ACCEPTED
ACCEPTED	NO	YES	UOW not restored; Status is restored	DISCARDED
ACCEPTED	NO	NO	UOW not restored; Status not restored	---
DELIVERED	YES	YES	UOW is restored; Status is restored	ACCEPTED
DELIVERED	YES	NO	UOW is restored; Status is restored	ACCEPTED

UOW Status before Restart	Persistent UOW: YES | NO	Persistent Status: YES | NO	Behavior of UOW and Status	UOW Status after Restart *
DELIVERED	NO	YES	UOW not restored; Status is restored	DISCARDED
DELIVERED	NO	NO	UOW not restored; Status not restored	---
PROCESSED **	YES	YES	Status is restored	PROCESSED
PROCESSED **	YES	NO	Status is not restored	---
PROCESSED **	NO	YES	Status is restored	PROCESSED
PROCESSED **	NO	NO	Status not restored	---

* If either UOW or its status is restored.

** In this state, the UOW information has already been deleted upon reaching PROCESSED status.

- **Restart Table 2**
  The behavior during restart of the following states does not depend on the settings of Persistent UOW; in these cases only the Persistent Status exists and does not change after a restart. There is no UOW to be restored.

UOW Status before Restart	Behavior of Status	UOW Status after Restart
CANCELLED	Status is restored	CANCELLED
DISCARDED	Status is restored	DISCARDED
BACKEDOUT	Status is restored	BACKEDOUT
TIMEDOUT	Status is restored	TIMEDOUT

Re-creation of Internal Control Blocks

To restore a UOW, the Broker re-creates all internal control blocks necessary to represent the UOW when it was accepted. The table displays the targets of each control block type:

Control Block Type	Association: Sender | Receiver	Notes
PCB	Sender; Receiver (optional)	PCB = Participant CB
SCB	Sender; Receiver	SCB = Service CB
CCB	Sender; Receiver	CCB = Conversation CB Two CCBs represent the conversation.
UWCB	Receiver	UWCB = unit of work CB The UWCB represents the UOW.

Note:
The messages associated with the UOW are not re-created in memory until a RECEIVE is actually issued for the UOW.

Behavior of Conversation at Broker Restart

Broker sets any units of work (UOWs) that are in DELIVERED status to ACCEPTED status during restart processing. If this is the first unit of work within a conversation sent by a client to a server, the assignment of the conversation to a particular server is dropped and the conversation is again available for all servers offering the same service.

If there is more than one unit of work in a single conversation and the first UOW is already received and committed by the server, the link to the server will kept even after this (non-first) UOW has reverted from DELIVERED to ACCEPTED status during restart processing. The server can retrieve units of work after restart with function RECEIVE OPTION=SYNC,CONVID=ANY and will get all old conversations containing UOWs first and then new conversations containing UOWs.

Servers performing a RECEIVE OPTION=SYNC, CONVID=NEW will retrieve only conversations not already assigned to this server. We strongly recommend that you implement RECEIVE OPTION=SYNC,CONVID=ANY or CONVID=OLD to retrieve already assigned conversations.

UOWs and Replicated Servers

Special consideration must be given when restarts occur, and there are persistent UOWs that are being sent to replicated servers, e.g. when more than one copy of a server is active. This is because a UOW is not associated with a server instance until the UOW's conversation is actually received by a server. From an application perspective, this means that a conversation that has not yet been received by its target server will be restored so that any instance of the server can process it. However, once the conversation has been received, any subsequent UOWs sent on the conversation will be restored so that only the specific instance, based on USER-ID and TOKEN, can receive them. The reasoning behind this is that a broker restart can occur without the servers being restarted, and the servers could be maintaining context information based on the conversation.

It is important to note that this can cause problems if the server instances are started as a result of load and the same load conditions are not present after the restart. For example, a UOW could be bound to the fifth instance of a server, but after a restart there is only enough load to start three instances. For this reason, we recommend that replicated servers using persistent UOWs not maintain any conversations with multiple UOWs.

Using Persistent Status

- When does Persistent Status make Sense?
- Adding Persistent Status to a UOW
- Resource and Performance Considerations

When does Persistent Status make Sense?

Persistent status should be considered for applications in which the sender needs to know if UOWs were actually processed successfully. In cases where the data associated with a UOW can be easily re-created in the event of a failure, persistent status may be a more desirable and lower-overhead alternative to a persistent UOW.

Persistent status does not require a persistent UOW.

Adding Persistent Status to a UOW

A UOW's status can be made persistent:

- by specifying a `UWSTATP` value between 1 and 254 in the ACI request that creates the UOW;
- by specifying a `UWSTATP` value between 1 and 254 in service definition or service defaults portion of the Broker attribute file, making the status of all UOWs for that service persistent; or
- by specifying a `UWSTATP` value between 1 and 254 in the Broker defaults section of the Broker attribute files, making the status of all UOWs in the system persistent.

Specifying `UWSTATP=255` in the ACI request that creates the UOW will explicitly make the UOW status non-persistent, overriding any broker or service default.

Resource and Performance Considerations

Using persistent status consumes resources in two areas.

- The persistent store is updated each time the UOW is modified, by either the sender or the receiver. These modifications occur whenever a `SEND` or `RECEIVE` function is issued for the UOW, or whenever its status is changed, such as by `SYNCPOINT OPTION=COMMIT`. Depending on the implementation, this will generate one or more I/O operations.
- The space used for the UOW (but not its messages) in the persistent store remains allocated for some period of time after the UOW has been completed.

The performance of individual requests will generally be affected by the additional time required to perform the I/O operations associated with maintaining persistent status. At this time, all operations are performed synchronously, although that may change in future releases.

Recovery Processing

- Introduction
- Determining the Status of a UOW
- A Real-world Example: Chess-by-Mail

Introduction

UOWs and persistence provide functionality for the application program (either client or server) to recover from failures: i.e., system, broker or application. In addition, this functionality allow new types of applications to be built, including ones not requiring concurrent execution of the client and server.

There are no standard rules for recovery, because each application model will use this functionality differently and will have different requirements for recovery. But the considerations in the following section should be kept in mind.

Determining the Status of a UOW

The most useful function for recovery is the `SYNCPOINT`, `OPTION=LAST`. This function will return the UOWID, CID, and status of the last UOW created by the caller, based on the `USER-ID` and `TOKEN`. This function can be used when an application starts or when it detects a failure to determine how much processing has been completed on a UOW. This information can then be used to decide how to recover from the failure.

A "Real-world" Example: Chess-by-Mail

Chess-by-mail is a sample of an application that takes advantage of UOWs, persistence, and persistent status. In generic terms, this application involves a client and a server exchanging messages on a single conversation. The conversation is long-running, and there is no requirement that the client and the server be active at the same time.

Although chess-by-mail was conceived as a single application, it is perhaps easier to describe its operation separately for the client and the server side. By convention, the white player is the client and the black player is the server. For simplicity, any user interaction has been left out of the description. Also for simplicity, only one chess-by-mail game is assumed to be running at any one time.

- Client Behavior
- Server Behavior

Client Behavior

The behavior of the chess-by-mail client is as follows:

1. Logon, specifying a `USER-ID` and `TOKEN`, which allow recovery of prior UOWs.
2. Issue `SYNCPOINT`, `OPTION`=`LAST` to determine the status of the last UOW created.
3. If the return code is `00780305 - UOW not found`, then there is no game in progress. So send the first white move to the server with: `SEND OPTION=COMMIT,CID=NEW`. If the send is successful, logoff and exit.
4. If the return code from `SYNCPOINT` is 0, then there is a last UOW and therefore a game is in progress. The UOW status value is examined to decide how to proceed.
5. If the status is `ACCEPTED`, then the server has not yet received the last move, so logoff and exit.
6. If the status is `DELIVERED`, then the server is currently processing the last move, so logoff and exit.
7. If the status is `TIMEOUT`, then the server did not receive the last move before its lifetime expired, so logoff and exit.
8. If the status is `PROCESSED`, then the server has received the last move and committed the UOW. Our application model has the client sending a move in response and committing both UOWs at the same time. So we need to receive the new move and send a reply to it.
9. Get the server's move with `RECEIVE,OPTION=SYNC,CID=`*n*, where *n* is the CID returned from `SYNCPOINT OPTION=LAST`.
10. Send the response move back using `SEND OPTION=SYNC,CID=n`.
11. Commit both the received and sent UOWs with a single call `SYNCPOINT OPTION=COMMIT,UOWID=BOTH`.
12. Logoff and exit.

Server Behavior

The behavior of the chess-by-mail server is as follows:

1. Logon, specifying a Userid and Token, which allow recovery of prior UOWs.
2. Register as the chess-by-mail server.
3. Issue `SYNCPOINT OPTION=LAST` to determine the status of the last UOW created.
4. If the return code is `00780305 - UOW not found`, then there is no game in progress. So we receive first white move from the client with: `RECEIVE OPTION=SYNC,CID=NEW`. When the `RECEIVE` has been completed, continue at step 11.
5. If the return code from `SYNCPOINT` is 0, then there is a last UOW and therefore a game is in progress. The UOW status value is examined to decide how to proceed.
6. If the status is `ACCEPTED`, then the client has not yet received the last move, so deregister, logoff and exit.
7. If the status is `DELIVERED`, then the client is currently processing the last move, so deregister, logoff and exit.
8. If the status is `TIMEOUT`, then the client did not receive the last move before its lifetime expired, so deregister, logoff and exit.
9. If the status is `PROCESSED`, then the client has received the last move and committed the UOW. Our application model has the server sending a move in response and committing both UOWs at the same time. So we need to receive the new move and send a reply to it.
10. Get the client's move with `RECEIVE,OPTION=SYNC,CID=`*n*, where *n* is the CID returned from

    `SYNCPOINT,OPTION=LAST`.
11. Send the response move back using `SEND,OPTION=SYNC,CID=`*n*.
12. Commit both the received and sent UOWs with a single call:

    `SYNCPOINT,OPTION=COMMIT,UOWID=BOTH`.
13. Deregister, logoff and exit.