Open Financial Exchange Specification 1.0.2

May 30, 1997

Chapters 5 – 7

Contents

5. INTERNATIONAL SUPPORT

5.1 LANGUAGE AND ENCODING
5.2 CURRENCY <CURDEF> <CURRENCY> <ORIGCURRENCY>
5.3 COUNTRY-SPECIFIC TAG VALUES

6. DATA SYNCHRONIZATION

6.1 OVERVIEW
6.2 BACKGROUND
6.3 DATA SYNCHRONIZATION APPROACH
6.4 DATA SYNCHRONIZATION SPECIFICS
6.5 CONFLICT DETECTION AND RESOLUTION
6.6 SYNCHRONIZATION VS. REFRESH
6.7 TYPICAL SERVER ARCHITECTURE FOR SYNCHRONIZATION
6.8 TYPICAL CLIENT PROCESSING OF SYNCHRONIZATION RESULTS
6.9 SIMULTANEOUS CONNECTIONS
6.10 SYNCHRONIZATION ALTERNATIVES
6.10.1 Lite Synchronization
6.10.2 Relating Synchronization and Error Recovery
6.11 EXAMPLES

7. FI PROFILE

7.1 OVERVIEW
7.1.1 Message Sets
7.1.2 Version Control
7.1.3 Batching and Routing
7.1.4 Client Signon for Profile Requests
7.1.5 Profile Request
7.2 PROFILE RESPONSE
7.2.1 Message Set
7.2.2 Signon Realms
7.2.3 Status Codes
7.3 PROFILE MESSAGE SET PROFILE INFORMATION

5International Support

5.1Language and Encoding

Most of the content in Open Financial Exchange is language-neutral. However, some error messages, balance descriptions, and similar tags contain text meant to appear to the financial institution customers. There are also cases, such as e-mail records, where customers need to send text in other languages. To support world-wide languages, Open Financial Exchange must identify the basic text encoding, character set, and language.

The Open Financial Exchange headers specify the encoding and character set, as described in Chapter 2. Current encoding values are USASCII and UNICODE. For USASCII, character set values are code pages. UNICODE ignores the character set *per se* although it still requires the syntax. Servers must respond with the encoding and character set requested by the client.

Clients identify the language in the signon request. Open Financial Exchange specifies languages by three-letter codes as defined in ISO-639. Servers report their supported languages in the profile (see Chapter 7). If a server cannot support the language requested by the client, it must return an error and not process the rest of the transactions.

5.2Currency <CURDEF> <CURRENCY> <ORIGCURRENCY>

In each transaction involving amounts, responses include a default currency identification, <CURDEF>. The values are based on the ISO-4217 three-letter currency identifiers.

Within each transaction, specific parts of the response might need to report a different currency. Where appropriate, aggregates include an optional <CURRENCY> aggregate. The scope of a <CURRENCY> aggregate is everything within the same aggregate that the <CURRENCY> aggregate appears in, including nested aggregates, unless overridden by a nested <CURRENCY> aggregate. For example, specifying a <CURRENCY> aggregate in an investment statement detail means that the unit price, transaction total, commission, and all other amounts are in terms of the given currency, not the default currency.

Note that there is no way for two or more individual elements that represent amounts—and are directly part of the same aggregate—to have different currencies. For example, there is no way in a statement download to have a different currency for the <LEDGERBAL> and the <AVAILBAL>, because they are both directly members of <STMTRS>. In most cases, you can use the optional <BAL> aggregates to overcome this limitation, since <BAL> aggregates accept individual <CURRENCY> aggregates.

The default currency for a request is the currency of the source account. For example, the currency for <BANKACCTFROM>.

The <CURRATE> should be the one in effect throughout the scope of the <CURRENCY> aggregate. It is not necessarily the current rate. Note that the <CURRATE> needs to take into account the choice of the FI for formatting of amounts (that is, where the decimal is) in both default and overriding currency, so that a client can do math. This can mean that the rate is adjusted by orders of magnitude (up or down) from what is commonly reported in newspapers.

Tag	*Description*
<CURRENCY> *or* **<ORIGCURRENCY>**	Currency aggregate
<CURRATE>	Ratio of <CURDEF> currency to <CURSYM> currency, in decimal form, *rate*
<CURSYM>	ISO-4217 3-letter currency identifier, *A-3*
</CURRENCY> *or* **</ORIGCURRENCY>**	

In some cases, Open Financial Exchange defines transaction responses so that amounts have been converted to the home currency. However, Open Financial Exchange allows FIs to optionally report the original amount and the original (foreign) currency. In these cases, transactions include a specific tag for the original amount, and then a <ORIGCURRENCY> tag to report the details of the foreign currency.

Again, <CURRENCY> means that Open Financial Exchange *has not* converted amounts. Whereas, <ORIGCURRENCY> means that Open Financial Exchange *has* already converted amounts.

5.3Country-Specific Tag Values

Some of the tags in Open Financial Exchange have values that are country-specific. For example, <USPRODUCTTYPE> is useful only within the United States. Open Financial Exchange will extend in each country as needed to provide tags that accept values useful to that country. Clients in other countries that do not know about these tags must simply skip them.

In some cases, a tag value represents a fundamental way of identifying something, yet there does not exist a world-wide standard for such identification. Examples include bank accounts and securities. In these cases, Open Financial Exchange must define a single, extensible approach for identification. For example, CUSIPs are used within the U.S., but not in other countries. However, CUSIPs are fundamental to relating investment securities, holdings, and transactions. Thus, a security ID consists of a two-part aggregate: one to identify the naming scheme, and one to provide a value. Open Financial Exchange will define valid naming schemes as necessary for each country.

6Data Synchronization

6.1Overview

Currently, some systems provide only limited support for error recovery and no support for backup files or multiple clients. The Open Financial Exchange data synchronization approach described in this chapter handles all of these situations.

Open Financial Exchange defines a powerful form of data synchronization between clients and servers.

Open Financial Exchange data synchronization addresses the following problems:

- Error recovery
- Use of multiple client applications
- Restoring from a backup file
- Multiple data files (for example, one copy at home, another at work).

This chapter first provides a brief background of error recovery problems and then presents the basic strategy used in Open Financial Exchange to perform data synchronization. Each Open Financial Exchange service includes specific details for data synchronization requests and responses.

Most of the information in this chapter concerns data synchronization, since it is a relatively new concept. The final section in this chapter discusses alternatives to full synchronization, and summarizes the options for each.

6.2Background

When a client begins a connection with a server for which the connection does not successfully complete, there are two main problems:

- Unconfirmed requests

  If a client does not receive a response to work it initiates, it has no way of knowing whether the server processed the request. It also does not have any server-supplied information about the request, such as a server ID number.

- Unsolicited data

  Some banking protocols allow a server to send data to the client whenever the client makes a connection. This specification assumes that the first client that calls in after the unsolicited data is available for download receives the data. If the connection fails, this information would be forever lost to the client. Examples of unsolicited data include updates in the status of a bill payment and e-mail responses.

Unsolicited data presents problems beyond error recovery. Because the first client that connects to a server is the only one to receive unsolicited data, this situation precludes use of multiple clients without a data synchronization method. For example, if a user has a computer at work and one at home, and wants to perform online banking from both computers, a bank server could send unsolicited data to one but not the other.

An even greater problem occurs when a user resorts to a backup copy of the client data file. This backup file will be missing recent unsolicited data with no way to retrieve it from the server once the server sends it.

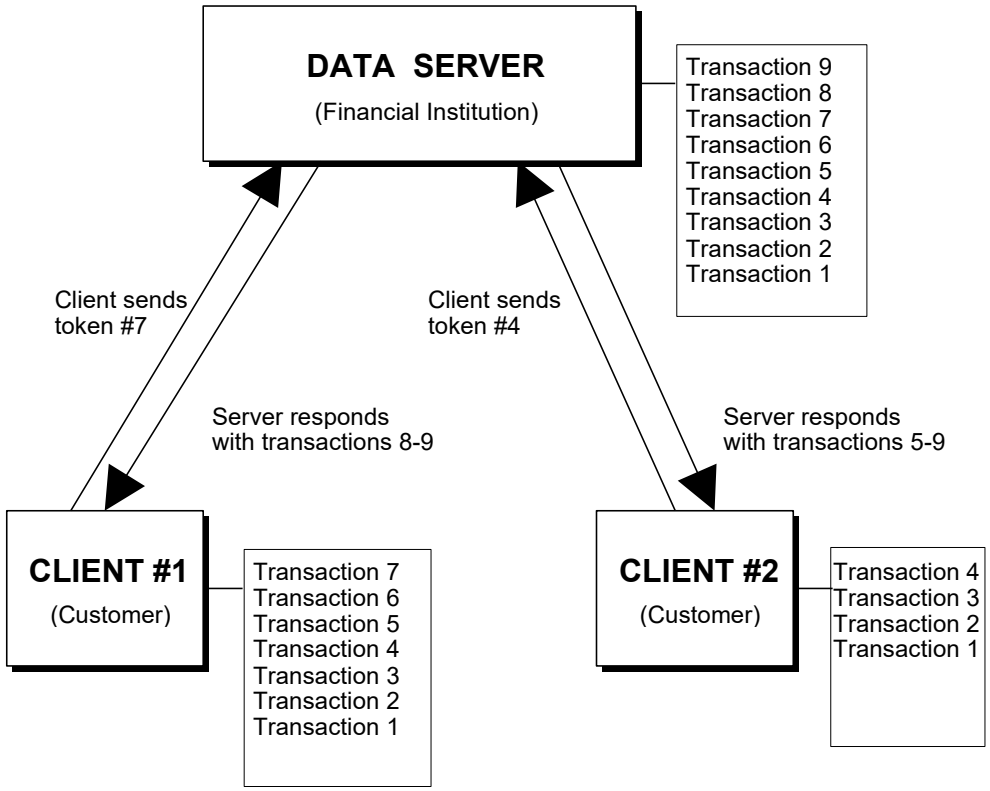
6.3Data Synchronization Approach

A simple solution is to make sure that clients can always obtain information from the server for a reasonable length of time. Clients can request recent responses—whether due to client-initiated work or other status changes on the server—by supplying the previous endpoint in the response history. Servers always supply a new endpoint whenever they supply responses.

If a client connection fails—or a client receives a response, but crashes before updating its database—the client will not save the new endpoint. On the next synchronization request, the server sends the same information (plus any further status changes).

If a user switches to a backup file, again the client will use the older endpoint in the synchronization request.

If multiple clients are in use, each will send requests based on its own endpoint, so that both clients will obtain complete information from the server. This is one reason why Open Financial Exchange responses carry enough information from the request to enable them to be processed on their own. The diagram below shows the relationship between clients and servers.



Open Financial Exchange relieves the server from maintaining any special error-recovery state information. However, Open Financial Exchange requires the server to maintain a history of individual responses and a way to identify a position in the history. This ID could be a time stamp, or be based on its existing state information.

> ***NOTE:*** *Open Financial Exchange does not require servers to store responses based on individual connections. Also, not all requests are subject to synchronization. For example, Open Financial Exchange does not require servers to store statement-download responses separately for data synchronization.*

6.4Data Synchronization Specifics

Open Financial Exchange does synchronization separately for each type of response. In addition, a synchronization request might include further identifying information, such as a specific account number. This specification defines the additional information for each synchronization request.

Each Open Financial Exchange service identifies the responses that are subject to data synchronization. For example, a bank statement-download is a read-only operation from the server. A client can request again if it fails; consequently, there is no data synchronization for this type of response.

The basis for synchronization is a *token* as defined by the <TOKEN> tag. The server can create a token in any way it wishes. The client simply holds the token for possible use in a future synchronization request.

The server can derive a token from one of the following:

- Time stamp
- Sequential number
- Unique non-sequential number
- Other convenient values for a server

> ***NOTE:*** *Open Financial Exchange reserves a <TOKEN> value of zero for the first time each type of response does a synchronization task.*

Clients will send a <TOKEN> of zero on their first synchronization request. Servers should send all available history, allowing a new client to know about work done by other clients. If a user's account has never been used with Open Financial Exchange, the server returns no history.

The server can use different types of tokens for different types of responses, if suitable for the server.

Tokens can contain up to 10 alphanumeric characters; see Chapter 3, "Common Aggregates, Elements, and Data Types." Tokens need to be unique only with respect to the type of synchronization request and the additional information in that request. For example, a bill payment synchronization request takes an account number; therefore, a token needs to be unique only within payments for the account.

Servers will not have infinite history available, so synchronization responses can optionally include a <LOSTSYNC> element with a value of Y (yes) if the old token in the synchronization request was older than the earliest available history. This tag allows clients to alert users that some responses have been lost.

NOTE: *Tokens are unrelated to <TRNUID>s, <SRVRTID>s, and <FITID>s, each of which serves a specific purpose and has its own scope and lifetime.*

A <SRVRTID> is not appropriate as a <TOKEN> for bill payment. A single payment has a single <SRVRTID>, but it can undergo several state changes over its life and thus have several entries in the token history.

There are three different ways a client and a server can conduct their requests and responses:

- Explicit synchronization – A client can request synchronization without sending any other Open Financial Exchange requests. The client sends a synchronization request, including the current token for that request. The response includes responses more recent than the given token, along with a new token.

- Synchronization with new requests – A client can request synchronization as part any new requests it has. The client gives the old token. The response includes responses to the new requests plus any others that became available since the old token, along with a new token. An aggregate contains the requests so that the server can process the new requests and update the token as an atomic action.

- New requests without synchronization – A client can make new requests without providing the old token. In this case, it expects just responses to the new requests. A subsequent request for synchronization will cause the server to send the same response again, because the client did not update its token.

Each request and response that requires data synchronization will define a synchronization aggregate. The aggregate tells the server which kind of data it should synchronize. By convention, these aggregates always have SYNC as part of their tag names, for example, <PMT**SYNC**RQ>. You can use these aggregates on their own to perform explicit synchronization, or as wrappers around one or more new transactions. For example, you can use <PMTSYNCRQ> aggregates to request synchronization in combination with new work. You can use the <PMTTRNRQ> by itself if you do not require synchronization.

Some clients can choose to perform an explicit synchronization before sending any new requests (with or without synchronization). This practice allows clients to be up-to-date and possibly interact with the user before sending any new requests. Other clients can simply send new requests as part of the synchronization request.

If a client synchronizes in one file, then sends new work inside a synchronization request in a second file, there is a small chance that additional responses become available between the two connections. There is even a smaller chance that these would be conflicting requests, such as modifications to the same object. However, some clients and some requests might require absolute control, so that the user can be certain that they are changing known data. To support this, synchronization requests can optionally specify <REJECTIFMISSING> element. The tag tells a server that it should reject all enclosed requests if the supplied <TOKEN> is out of date *before considering the new requests.* That is, if any new responses became available, whether related to the incoming requests or not (but part of the scope of the synchronization request), the server should immediately reject the requests. It should still return the new responses. A client can then try again until it finds a stable window to submit the work. See section 6.5 for more information about conflict detection and resolution.

The password change request and response present a special problem. See section 2.5.2 for further information.

6.5Conflict Detection and Resolution

Conflicts arise whenever two or more users or servers modify the same data. This can happen to any object that has a <SRVRTID> that supports change or delete requests. For example, one spouse and the other might independently modify the same recurring bill payment model. From a server perspective, there is usually no way to distinguish between the same user making two intended changes and two separate users making perhaps unintended changes. Therefore, Open Financial Exchange provides enough tools to allow clients to carefully detect and resolve conflicts. Open Financial Exchange requires only that a server process atomically all requests in a single <OFX> block.

A careful client always synchronizes before sending any new requests. If any responses come back that could affect a user's pending requests, the client can ask the user whether it should still send those pending requests. Because there is a small chance for additional server actions to occur between the initial synchronization request and sending the user's pending requests, extremely careful clients can use the <REJECTIFMISSING> element. Clients can iterate sending pending requests inside a synchronization request with <REJECTIFMISSING> and testing the responses to see if they conflict with pending requests. A client can continue to do this until a window of time exists wherein the client is the only agent trying to modify the server. In reality, this will almost always succeed on the first try.

6.6Synchronization vs. Refresh

There are some situations, and some types of clients, for which it is preferable that the client ask the server to send everything it knows, rather than just a set of changes. For example, a client that has not connected often enough may have lost synchronization. Or, the user might be using a completely stateless client, such as a web browser.

> ***NOTE:*** *Open Financial Exchange does not require a client to refresh just because it has lost synchronization.*

Clients will mainly want to refresh lists of long-lived objects on the server; generally objects with a <SRVRTID>. For example, Open Financial Exchange Payments has both individual payments and models of recurring payments.

A brand new client, or a client that lost synchronization, might want to learn about in-progress payments by doing a synchronization refresh of the payment requests. It would almost certainly want to do a synchronization refresh of the recurring payment models, because these often live for months or years.

A client might not perform a synchronization refresh on e-mail responses.

A client can request a refresh by using the <REFRESH> tag with value of Y instead of the <TOKEN> tag. Server descriptions detail the exact behavior that servers should follow. However, the general rule is that servers should send responses that emulate a client creating or adding each of the objects governed by the particular synchronization request.

In these cases, servers set <TRNUID> to zero; the standard value for server-generated responses.

There is no need to recreate a stream of responses that emulate the entire history of the object, just an add response that reflects the current state. For example, if you create a model and then modify it three times, even if this history would have been available for a regular synchronization, servers should only send a single add that reflects the current state.

A client that wants only the current token, without refresh or synchronization, makes requests with <TOKENONLY> and a value of Y.

In all cases, servers should send the current ending <TOKEN> for the synchronization request in refresh responses. This allows a client to perform regular synchronization requests in the future.

The following table summarizes the options in a client synchronization request:

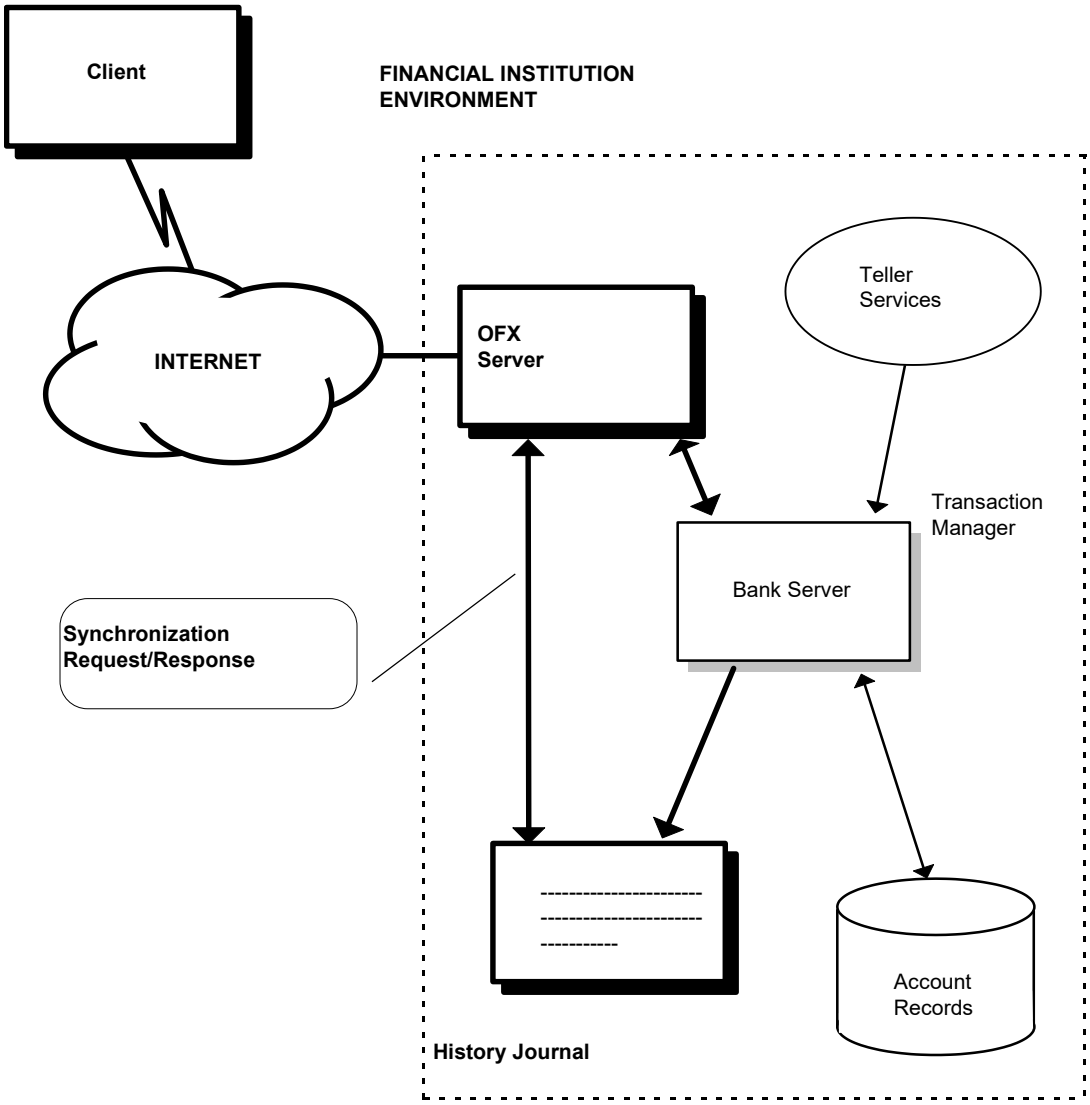
Tag	*Description*
Client synchronization option; <TOKEN>, <TOKENONLY>, or <REFRESH>	
<TOKEN>	Previous value of <TOKEN> received for this type of synchronization request from server; 0 for first-time requests; *token*
<TOKENONLY>	Request for just the current <TOKEN> without the history, *Boolean*
<REFRESH>	Request for refresh of current state, *Boolean*
<REJECTIFMISSING>	If Y, do not process requests if client <TOKEN> is out of date, *Boolean*

6.7Typical Server Architecture for Synchronization

This section describes how an FI can approach supporting synchronization based on the assumption that modifications to an existing financial server will be kept to a minimum.

The simplest approach is to create a history database separate from the existing server. This history could consist of the actual Open Financial Exchange transaction responses (<TRNRS> aggregates) that are available to a synchronization request. The history database could index records by token, response type, and any other identifying information for that type, such as account number.

The diagram below shows a high-level model of the Open Financial Exchange architecture for a financial institution. Notice that the diagram shows the presence of a history journal.



The server adds responses to the history journal for any action that takes place on the existing server. This is true whether the Open Financial Exchange requests initiate the action or, in the case of recurring payments, it happens automatically on the server. Once added to the history journal, the server can forget them.

The areas of the Open Financial Exchange server that process synchronization requests need only search this history database for matching responses that are more recent than the incoming token.

For a refresh request, an Open Financial Exchange server would access the actual bank server to obtain the current state rather than recent history.

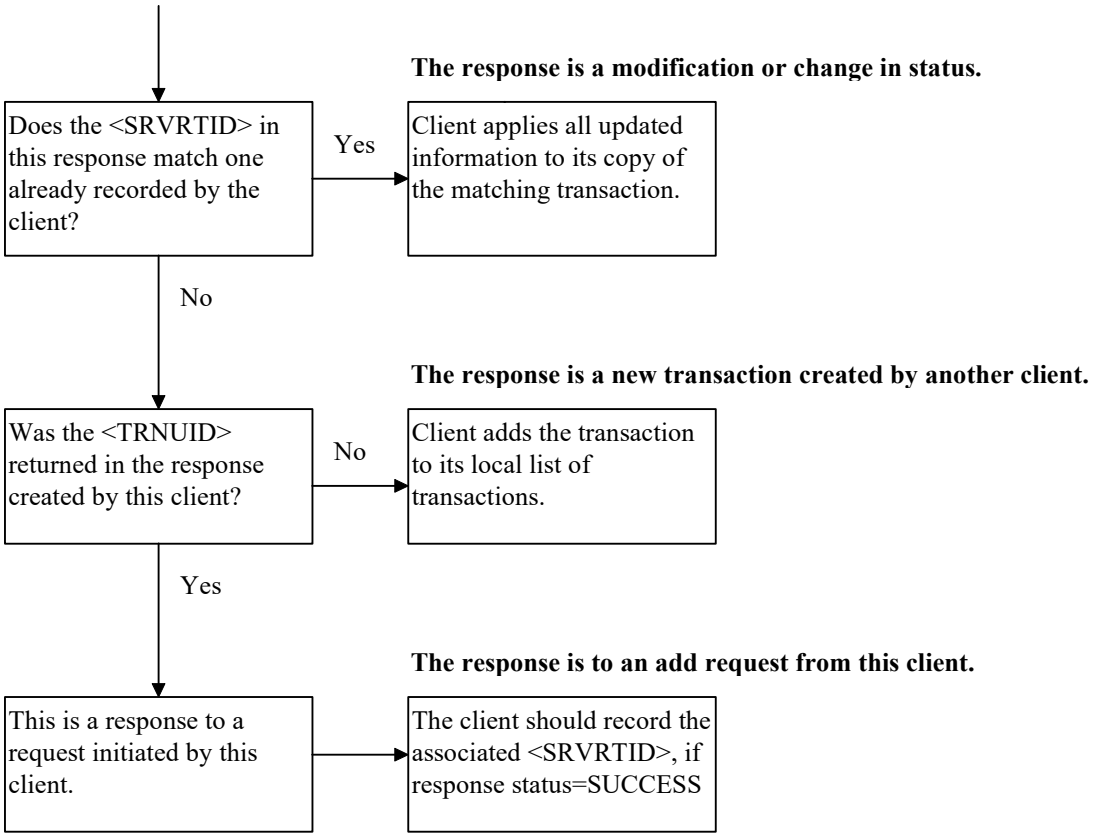
Periodically the bank server would purge the history server of older entries.

Only requests that are subject to synchronization need to have entries in the history database. Statement downloads do not involve synchronization; therefore, the FI server should not add these responses to the history database. Since statement downloads are usually the largest in space and the most frequent, eliminating these saves much of the space a response history might otherwise require.

More sophisticated implementations can save even more space. The history database could save responses in a coded binary form that is more compact than the full Open Financial Exchange response format. Some FIs might have much or all of the necessary data already in their servers; consequently, they would not require new data. An FI could regenerate synchronization responses rather than recall them from a database.

6.8Typical Client Processing of Synchronization Results

The diagram below shows a general flowchart of what an Open Financial Exchange client would do with the results of a synchronization request. Most requests and responses subject to data synchronization contain both <TRNUID> and <SRVRTID>.



6.9Simultaneous Connections

It is increasingly common that a server can get simultaneous or overlapping requests from the same user over two different computers. Open Financial Exchange requires a server to process each set of requests

sent in a file as an atomic action. Servers can deal with the problems that arise with simultaneous use in two ways:

- Allow simultaneous connections, ensure each is processed atomically, and use the data synchronization mechanism to bring the two clients up to date. This is the preferred method.
- Lock out all but one user at a time, returning the error code 15501 for multiple users.

6.10Synchronization Alternatives

Although it is **RECOMMENDED** that Open Financial Exchange servers implement full synchronization as described in this chapter, an alternate approach, "lite synchronization," could be easier for some servers to support. This approach focuses only on error recovery and does not provide any support for multiple clients, multiple data files, or use of backup files. The approach is to preserve the message sets while simplifying the implementation.

In addition, some clients might prefer to use response-file based error recovery with all servers, even if the client and some servers support full synchronization. This section first describes lite synchronization, and then explains the rules that clients and servers use to decide how to communicate.

6.10.1Lite Synchronization

Lite synchronization requires servers to accept all synchronization messages, but does not require them to keep any history or tokens. Responses need to be sent only once and then the server can forget them. Responses to client requests, whether or not they are made inside a synchronization request, are processed normally. Responses that represent server-initiated work, such as payment responses that arise from recurring payments, are sent only in response to synchronization requests. A server does not have to hold responses in case a second client makes a synchronization request.

Because full synchronization supports error recovery, an alternative is needed for lite synchronization. Servers using lite synchronization keep a copy of the entire response file they last sent. Clients requesting that servers prepare for error recovery generate a globally unique ID for each file they send. In the OFX headers, there are two tags associated with error recovery:

- OLDFILEUID – UID of the last request and response that was successfully received and processed by the client
- NEWFILEUID – UID of the current file

The format of these is the same as used with <TRNUID> as documented in Section, 2.4.6.

Servers use the following rules:

- If OLDFILEUID is absent, the client is not requesting file-based error recovery for this session. The server does not need to save the response file. In addition, since the client is not specifying a previous file that can now be committed, the server should not search for a response file to delete. Optionally, a server can choose to test for a client error by checking whether NEWFILEUID matches a previous request file. If NEWFILEUID matches a previous request file, but the client did not send an OLDFILEUID, the client has committed an error. In this case, the server should report a general error.
- If the client sent OLDFILEUID and NEWFILEUID, and NEWFILEUID matches a previous request file, the client is requesting error recovery. The server should send the matching saved response file.

- If the client sent OLDFILEUID and NEWFILEUID, and OLDFILEUID matches a file saved on the server, then OLDFILEUID is a file that the client has successfully processed and the server can delete it. The client is also requesting that the response for the current file be saved under the NEWFILEUID for possible error recovery.

A server will never need to save more than one file per client data file, but because of possible multi-client or multi-data file usage, it might need to save several files for a given user. A server should save files for as long as possible, but not indefinitely. A server cannot recognize an error recovery attempt if it comes after it has purged the error recovery file. A server would process it as a new request. In this case, a server should recognize duplicate transaction UIDs for client-initiated work, such as payments, and then reject them individually. Server-generated responses would be lost to the client.

For a server accustomed to sending unsolicited responses, lite synchronization should closely match the current response-file based implementation. The only difference is that a server should hold the unsolicited responses until the client makes the first appropriate synchronization request; rather than automatically adding them to any response file. Once added, the server can mark them as delivered, relying on error recovery to insure actual delivery.

6.10.2Relating Synchronization and Error Recovery

Client and server developers should first decide whether or not they will support full synchronization. If they can, then they can support response-file error recovery as well, or they can rely on synchronization to perform error recovery. If they adopt only lite synchronization, Open Financial Exchange requires response-file error recovery. A server describes each of these choices in its server profile records. The following combinations are valid:

- Full synchronization with response-file error recovery
- Full synchronization without separate response-file error recovery
- Lite synchronization with response-file error recovery

Clients request response-file error recovery by including the old and new session UIDs in the header. If they are absent, servers need not save the response file for error recovery. Clients request synchronization by using those synchronization requests defined throughout this specification.

6.11Examples

Here is an example of full synchronization using bill payment as the service. Open Financial Exchange Payments provides two different synchronization requests and responses, each with their own token; one for payment requests and one for repeating payment model requests. Note that these simplified examples do not include the outer <OFX> layer, <SIGNON>, and so forth.

Client A requests synchronization:

```
<PMTSYNCRQ>
   <TOKEN>123
   <REJECTIFMISSING>N
   <BANKACCTFROM>
      <BANKID>121000248
      <ACCTID>123456789
      <ACCTTYPE>CHECKING
   </BANKACCTFROM>
</PMTSYNCRQ>
```

The server sends in response:

```
<PMTSYNCRS>
   <TOKEN>125
   <LOSTSYNC>N
   <BANKACCTFROM>
      <BANKID>121000248
      <ACCTID>123456789
      <ACCTTYPE>CHECKING
   </BANKACCTFROM>
   <PMTTRNRS>
      <TRNUID>123
      <STATUS>
         ... status details
      </STATUS>
      <PMTRS>
         ... details on a payment response
      </PMTRS>
   </PMTTRNRS>
   <PMTTRNRS>
      <TRNUID>546
      <STATUS>
         ... status details
      </STATUS>
      <PMTRS>
         ... details on another payment response
      </PMTRS>
   </PMTTRNRS>
</PMTSYNCRS>
```

Client A was missing two payment responses, which the server provides. At this point, client A is synchronized with the server. Client A now makes a new payment request, and includes a synchronization update as part of the request. This update avoids having to re-synchronize the expected response at a later time.

```
<PMTSYNCRQ>
   <TOKEN>125
   <REJECTIFMISSING>N
   <BANKACCTFROM>
```

```
      <BANKID>121000248
      <ACCTID>123456789
      <ACCTTYPE>CHECKING
   </BANKACCTFROM>
   <PMTTRNRQ>
      <TRNUID>12345
      <PMTRQ>
         ... details of a new payment request
      </PMTRQ>
   </PMTTRNRQ>
</PMTSYNCRQ>
```

The response to this new request:

```
<PMTSYNCRS>
   <TOKEN>126
   <LOSTSYNC>N
   <BANKACCTFROM>
      <BANKID>121000248
      <ACCTID>123456789
      <ACCTTYPE>CHECKING
   </BANKACCTFROM>
   <PMTTRNRS>
      ... details on a payment response to the new request
   </PMTTRNRS>
</PMTSYNCRS>
```

The client now knows that the server has processed the payments request it just made, and that nothing else has happened on the server since it last synchronized with the server.

Assume client B was synchronized with respect to payments for this account up through token 125. If it called in now and synchronized—with or without making additional requests—it would pick up the payment response associated with token 126. It records the same information that was in client A, which would give both clients a complete picture of payment status.

7FI Profile

7.1Overview

Open Financial Exchange clients use the profile to learn the capabilities of an Open Financial Exchange server. This information includes general properties such as account types supported, user password requirements, specific messages supported, and how the client should batch requests and where to send the requests. A client obtains a portion of the profile when a user first selects an FI. The client obtains the remaining information prior to sending any actual requests to that FI. The server uses a time stamp to indicate whether the server has updated the profile, and the client checks periodically to see if it should obtain a new profile.

In more detail, a profile response contains the following sections, which a client can request independently:

- Message Sets – list of services and any general attributes of those services. Message sets are collections of messages that are related functionally. They are generally subsets of what users see as a service.
- Signon realms – FIs can require different signons (user ID and/or password) for different message sets. Because there can only be one signon per <OFX> block, a client needs to know which signon the server requires and then provide the right signon for the right batch of messages.

The profile message is itself a message set. In files, Open Financial Exchange uses the <PROFMSGSV1> aggregate to identify this profile message set.

The following sections describe the general use of profile information.

7.1.1Message Sets

A message set is a collection of related messages. For example, Chapter 11, “Banking,” defines several message sets: statement download, credit card statement download, intrabank transfers, and so forth. A server routes all of the messages in a message set to a single URL and merges their versions together.

Clients and servers generally use message sets as the granularity to decide what functionality they will support. A “banking” server can choose to support the statement download and intrabank transfer message sets, but not the wire transfer message set. Attributes are available in many cases to further define how Open Financial Exchange supports a message set.

Each portion of the Open Financial Exchange specification that defines messages also defines the message set to which that the messages belongs. This includes what additional attributes are available for those messages, and whether Open Financial Exchange requires the message set or it is optional.

7.1.2Version Control

Message sets are the basis of version control. Over time there will be new versions of the message sets, and at any given time servers will likely want to support more than one version of a message set. Clients should also be capable of supporting as many versions as possible. Through the profile, clients discover which versions are supported for each message set. Clients and servers exchange messages at the highest common level for each message set.

For the Open Financial Exchange-SGML data format, there is a single DTD for all message sets. The version number of the DTD advances when any *syntactic* change is made to any of the message sets. (It is possible to make a *semantic* change that would not even require a change in syntax. A change in rules, for example, that would change the version of the message set without changing the DTD.) A single DTD cannot have two different definitions for the same aggregate. There are limitations to how a server that uses true DTD-based parsing can handle multiple versions of a message at the same time.

7.1.3Batching and Routing

To allow FIs to set up different servers for different message sets, different versions, or to directly route some messages to third party processors, message sets define the URL to which a server sends messages in that message set. Each version of a message set can have a different URL. In the common case where many or all message sets are sent to a single URL, clients will consolidate messages across compatible message sets. Clients can consolidate when:

- Message sets have the same URL
- Message sets have a common security level
- Message sets have the same signon realm

NOTE: *The signon message set can be used with other message sets even if it contains incompatible settings for the URL, security level, or signon realm. The message set information for signon messages is used only if the signon message is sent by itself.*

7.1.4Client Signon for Profile Requests

Clients must include a signon request <SONRQ> with every message, including profile requests. The first time that a client requests the FI profile, the signon request will be present, but the user ID and password will not be valid and will be ignored by the server.

NOTE: *Since elements cannot be set to a blank value, the user ID and password must contain at least one character*.

Once the user has enrolled and received his or her user ID and password, the client will request the profile again, even if the profile is not yet out-of-date. At this point, the server can respond with a profile response that indicates that the profile is up-to-date. Alternatively, if the FI wants to return a customer-specific profile, the FI can choose to return a new FI profile in the response.

For more information about signon requests, refer to section 2.5.

7.1.5Profile Request

A profile request indicates which profile components a client desires. It also indicates what the client's routing capability is. Profiles returned by the FI must be compatible with the requested routing style, or the server returns an error.

Profile requests are not subject to synchronization.

Profile requests must appear within a <PROFTRNRQ> transaction wrapper.

Tag	*Description*
<PROFRQ>	Profile-request aggregate
<CLIENTROUTING>	Identifies client routing capabilities, see table below
<DTPROFUP>	Date and time client last received a profile update, *datetime*
</PROFRQ>	

Tag	*Description*
NONE	Client cannot perform any routing. All URLs must be the same. All message sets share a single signon realm.
SERVICE	Client can perform limited routing. See details below.
MSGSET	Client can route at the message-set level. Each message set can have a different URL and/or signon realm.

The SERVICE option supports clients that can route bill payment messages to a separate URL from the rest of the messages. Because the exact mapping of message sets to the general concept of bill payment can vary by client and by locale, this specification does not provide precise rules for the SERVICE option. Each client will define its requirements.

7.2Profile Response

To determine whether the client has the latest version of the FIs profile, the server checks the date and time passed by the client in <DTPROFUP>.

If the client has the latest version of the FIs profile, the server returns status code 1 in the <STATUS> aggregate of the profile-transaction aggregate <PROFTRNRS>. The server does not return a profile-response aggregate <PROFRS>.

If the client does not have the latest version of the FI profile, the server responds with the profile-response aggregate <PROFRS> in the profile-transaction aggregate <PROFTRNRS>. The response includes message set descriptions, signon information, and general contact information.

Tag	*Description*
<PROFRS>	Profile-response aggregate
<MSGSETLIST>	Beginning list of message set information
<XXXMSGSET>	One or more message set aggregates
</XXXMSGSET>	
</MSGSETLIST>	
<SIGNONINFOLIST>	Beginning of signon information
<SIGNONINFO>	One or more signon information aggregates
</SIGNONINFO>	
</SIGNONINFOLIST>	
<DTPROFUP>	Time this was updated on server, *datetime*
<FINAME>	Name of institution, *A-32*
<ADDR1>	FI address, line 1, *A-32*
<ADDR2>	FI address, line 2, *A-32*
<ADDR3>	FI address, line 3, *A-32*
<CITY>	FI address city, *A-32*
<STATE>	FI address state, *A-5*
<POSTALCODE>	FI address postal code, *A-11*
<COUNTRY>	FI address country; 3-letter country code from ISO/DIS-3166, *A-3*
<CSPHONE>	Customer service telephone number, *A-32*
<TSPHONE>	Technical support telephone number, *A-32*
<FAXPHONE>	Fax number, *A-32*
<URL>	URL for general information about FI (not for sending data), *URL*
<EMAIL>	E-mail address for FI, *A-80*
</PROFRS>	

7.2.1Message Set

An aggregate describes each message set supported by an FI. Message sets in turn contain an aggregate for each version of the message set that is supported. For a message set named *XXX*, the convention is to name the outer aggregate <*XXX*MSGSET> and the tag for each version <*XXX*MSGSETVn>.

The reason for message set-specific aggregates is that the set of attributes depends on the message set. These can change from version to version, so there are version-specific aggregates as well.

The general form of the response is:

Tag	*Description*
<XXXMSGSET>	Service aggregate
<XXXMSGSETVn>	Version-of-message-set aggregate, 1 or more
</XXXMSGSETVn>	
</XXXMSGSET>	

The <***XXX*MSGSETVn**> aggregate has the following form:

Tag	*Description*
<*XXX*MSGSETVn>	Message-set-version aggregate
<MSGSETCORE>	Common message set information
</MSGSETCORE>	
message-set specific	Zero or more attributes specific to this version of this message set, as defined by each message set
</*XXX*MSGSETVn>	

The common message set information <MSGSETCORE> is as follows:

Tag	*Description*
<MSGSETCORE>	Common-message-set-information aggregate
<VER>	Version number of the message set, (for example, <VER>1 for version 1 of the message set), *N-5*
<SPNAME>	Service provider name, *A-32* *Some financial institutions out-source their OFX servers to a service provider. In such cases, the SPNAME element should be included in the MSGSETCORE.*
<URL>	URL where messages in this set are to be sent, *URL*
<OFXSEC>	Security level required for this message set; see Chapter 4. *NONE or TYPE 1.*
<TRANSPSEC>	Y if transport-level security must be used, N if not used; see Chapter 4. *Boolean*
<SIGNONREALM>	Signon realm to use with this message set, *A-32*
<LANGUAGE>	Language supported, *language.* If more than one language is supported, multiple <Language> tags can be sent.
<SYNCMODE>	FULL for full synchronization capability LITE for lite synchronization capability See Chapter 6 for more information.
<RESPFILEER>	Y if server supports response-file based error recovery, *Boolean* See Chapter 6 for more information.
</MSGSETCORE>	

NOTE: For all message sets currently defined in Open Financial Exchange, the value of <TRANSPSEC> must be YES.

NOTE: Within a <MSGSETCORE> aggregate, the <VER> element defines the version number of that message set. It does not refer to the version number of the Open Financial Exchange specification or the DTD files. For more information about message sets and version numbers, refer to section 2.4.5.

7.2.2Signon Realms

A signon realm identifies a set of messages that can be accessed using the same password. Realms are used to disassociate signons from specific services, allowing FIs to require different signons for different message sets. In practice, FIs will want to use the absolute minimum number of realms possible to reduce

the user's workload.

Tag	*Description*
<SIGNONINFO>	Signon-information aggregate
<SIGNONREALM>	Identifies this realm, *A-32*
<MIN>	Minimum number of password characters, *N-2*
<MAX>	Maximum number of password characters, *N-2*
<CHARTYPE>	Type of characters allowed in password; one of ALPHAONLY, NUMERICONLY, ALPHAORNUMERIC, or ALPHAANDNUMERIC.
<CASESEN>	Y if password is case-sensitive, *Boolean*
<SPECIAL>	Y if special characters are allowed, *Boolean*
<SPACES>	Y if spaces are allowed, *Boolean*
<PINCH>	Y if server supports USERPASSPIN-change requests, *Boolean*
<CHGPINFIRST>	Y if server requires clients to change USERPASSPIN as part of first signon, *Boolean*
</SIGNONINFO>	

7.2.3Status Codes

Value	*Meaning*
0	Success (INFO)
1	Client is up-to-date (INFO)
2000	General error (ERROR)

7.3Profile Message Set Profile Information

The profile message set functions the same way as all other message sets; therefore, it contains a profile description for that message set. Because <PROFMSGSET> is always part of a message set response, it is described here. Servers must include the <PROFMSGSET> as part of the profile response <MSGSETLIST>. There are no attributes, but the aggregate must be present to indicate support for the message set.

Tag	*Description*
<PROFMSGSET>	Message-set-profile-information aggregate
<PROFMSGSETV1>	Opening tag for V1 of the message set profile information
<MSGSETCORE>	Common message set information
</MSGSETCORE>	
</PROFMSGSETV1>	
</PROFMSGSET>	