When to load CSS

Large HTML page with Images

This page shall test if the recorder generates scripts with requests in correct order. It includes images in sequential order: stadyn_image1.gif through stadyn_image10.gif

---

stadyn_image1

stadyn_image2

Open Financial Exchange Specification 1.0

February 14, 1997

Chapters 1 - 10

stadyn_image3

stadyn_image4

stadyn_image5

Contents

1. Overview 51.1 Introduction 51.1.1 Design Principles 51.2 Open Financial Exchange at a Glance 71.2.1 Data Transport 71.2.2 Request and Response Model 81.3 Conventions 92. Structure 102.1 HTTP Headers 102.2 Open Financial Exchange Headers 112.2.1 The Meaning of Version Numbers 122.3 SGML Details 122.3.1 Compliance 122.3.2 Special Characters 122.4 Open Financial Exchange SGML Structure 132.4.1 Overview 132.4.2 Top Level 132.4.3 Messages 132.4.4 Message Sets and Version Control 142.4.5 Transactions 152.5 The Signon Message Set 162.5.1 Signon <SONRQ> <SONRS> 162.5.2 PIN Change <PINCHRQ> <PINCHRS> 192.5.3 Examples 202.6 External Data Support 202.7 Extensions to Open Financial Exchange 213. Common Aggregates, Elements, and Data Types 223.1 Common Aggregates 223.1.1 Identifying Financial Institutions and Accounts 223.1.2 Balance Records <BAL> 223.1.3 Error Reporting <STATUS> 233.2 Common Elements 243.2.1 Financial Institution Transaction ID <FITID> 243.2.2 Server-Assigned ID <SRVRTID> 243.2.3 Client-Assigned Transaction

UID <TRNUID> 253.2.4 Token <TOKEN> 253.2.5 Transaction Amount <TRNAMT> 253.2.6 Memo <MEMO> 253.2.7 Date Start and Date End <DTSTART> <DTEND> 263.3 Common data types 263.3.1 Dates and Times 263.3.2 Amounts, Prices, and Quantities 283.3.3 Language 283.3.4 Basic data types 284. Security 294.1 Security Solutions 294.1.1 Determining Security Levels <OFXSEC> <TRANSPSEC> 294.2 Channel-Level Security 304.2.1 Security Requirements 304.2.2 Using SSL 3.0 in Open Financial Exchange 304.3 Application-Level Security 314.3.1 Requirements for Application-Layer Security 314.3.2 Using Application-level Encryption in Open Financial Exchange 325. International Support 335.1 Language and Encoding 335.2 Currency <CURDEF> <CURRENCY> <ORIGCURRENCY> 335.3 Country-Specific Tag Values 346. Data Synchronization 356.1 Overview 356.2 Background 356.3 Data Synchronization Approach 366.4 Data Synchronization Specifics 376.5 Conflict Detection and Resolution 396.6 Synchronization vs. Refresh 406.7 Typical Server Architecture for Synchronization 416.8 Typical Client Processing of Synchronization Results 436.9 Simultaneous Connections 446.10 Synchronization Alternatives 446.10.1 Lite Synchronization 446.10.2 Relating Synchronization and Error Recovery 456.11 Examples 467. FI Profile 487.1 Overview 487.1.1 Message Sets 487.1.2 Version Control 497.1.3 Batching and Routing 497.2 Profile Request 507.3 Profile Response 517.3.1 Message Set 527.3.2 Signon Realms 537.3.3 Status Codes 537.4 Profile Message Set Profile Information 548. Activation & Account Information 558.1 Overview 558.2 Approaches to User Sign-Up with Open Financial Exchange 558.3 Users and Accounts 568.4 Enrollment and Password Acquisition <ENROLLRQ> <ENROLLRS> 568.4.1 User IDs 578.4.2 Enrollment Request 578.4.3 Enrollment Response 598.4.4 Enrollment Status Codes 598.4.5 Examples 608.5 Account Information 608.5.1 Request <ACCTINFORQ> 618.5.2 Response <ACCTINFORS> 618.5.3 Account Information Aggregate <ACCTINFO> 628.5.4 Status Codes 628.5.5 Examples 638.6 Service Activation 638.6.1 Activation Request and Response 648.6.2 Service Activation Synchronization 668.6.3 Examples 668.7 Name and Address Changes <CHGUSERINFORQ> <CHGUSERINFORS> 678.7.1 <CHGUSERINFORQ> 678.7.2 <CHGUSERINFORS> 688.7.3 Status Codes 688.8 Signup Message Set Profile Information 699. Customer to FI Communication 709.1 The E-Mail Message Set 709.2 E-Mail Messages 709.2.1 Regular vs. Specialized E-Mail 719.2.2 Basic <MAIL> Aggregate 719.2.3 E-Mail <MAILRQ> <MAILRS> 719.2.4 E-Mail Synchronization <MAILSYNCRQ> <MAILSYNCRS> 729.2.5 Example

739.3 Get HTML Page 749.3.1 MIME Get Request and Response <GETMIMERQ> <GETMIMERS> 749.3.2 Example 759.4 E-Mail Message Set Profile Information 7610. Recurring Transactions 7710.1 Creating a Recurring Model 7710.2 Recurring Instructions <RECURRINST> 7710.2.1 Values for <FREQ> 7810.2.2 Examples 7910.3 Retrieving Transactions Generated by a Recurring Model 8010.4 Modifying and Canceling Individual Transactions 8010.5 Modifying and Canceling Recurring Models 8010.5.1 Examples 81

1. Overview

2. Introduction

Open Financial Exchange is a broad-based framework for exchanging financial data and instructions between customers and their financial institutions. It allows institutions to connect directly to their customers without requiring an intermediary.

Open Financial Exchange is an open specification that anyone can implement: any financial institution, transaction processor, software developer or other party. It uses widely accepted open standards for data formatting (such as SGML), connectivity (such as TCP/IP and HTTP), and security (such as SSL).

Open Financial Exchange defines the request and response messages used by each financial service as well as the common framework and infrastructure to support the communication of those messages. This specification does not describe any specific product implementation.

1. Design Principles

The following principles were used in designing Open Financial Exchange:

l **Broad Range of Financial Activities** - Open Financial Exchange provides support for a ***broad*** range of financial activities. Open Financial Exchange 1.0 specifies the following services:

n Bank statement download

n Credit card statement download

n Funds transfers including recurring transfers

n Consumer payments, including recurring payments

n Business payments, including recurring payments

n Brokerage and mutual fund statement download, including transaction history, current holdings and balances

l **Broad Range of Financial Institutions** - Open Financial Exchange supports communication with a ***broad*** range of financial institutions (FIs), including:

n Banks

n Brokerage houses

n Merchants

n Processors

n Financial advisors

n Government agencies

l **Broad Range of Front-End applications** - Open Financial Exchange supports a ***broad*** range of front-end applications covering all types of financial activities running on all types of platforms, including Web-based applications.

l **Extensible** - Open Financial Exchange has been designed to allow the easy addition of new services. Future versions will include support for many new services.

l **Open** - This specification is publicly available. You can build client and server applications using the Open Financial Exchange protocols independent of any specific technology, product, or company.

l **Multiple Client Support** - Open Financial Exchange allows a user to use multiple client applications to access the same data at a financial institution. With the popularity of the World Wide Web, customers are increasingly more likely to use multiple applications-either desktop-based or Web-based-to perform financial activities. For example, a customer can track personal finances at home with a desktop application and occasionally pay bills while at work with a Web-based application. The use of data synchronization to support multiple clients is a key innovation in Open Financial Exchange.

l **Robust** - Open Financial Exchange will be used for executing important financial transactions and for communicating important financial information. Assuring users that transactions are executed and information is correct is crucial. Open Financial Exchange provides robust protocols for error recovery.

l **Secure** - Open Financial Exchange provides a framework for building secure online financial services. In Open Financial Exchange, security encompasses authentication of the parties involved, as well as secrecy and integrity of the information being exchanged.

l **Batch & Interactive** - The design of request and response messages in Open Financial Exchange is for use in either batch or interactive style of communication. Open Financial Exchange provides for applying a single authentication context to multiple requests in order to reduce the overhead of user authentication.

l **International Support** - Open Financial Exchange is designed to supply financial services throughout the world. It supports multiple currencies, country-specific extensions, and different forms of encoding such as UNICODE.

l **Platform Independent** -Open Financial Exchange can be implemented on a wide variety of front-end client devices, including those running Windows 3.1, Windows 95, Windows NT, Macintosh, or UNIX. It also supports a wide variety of Web-based environments, including those using HTML, Java, JavaScript, or ActiveX. Similarly on the back-end, Open Financial Exchange can be implemented on a wide variety of server systems, including those running UNIX, Windows NT, or OS/2.

l **Transport Independent** - Open Financial Exchange is independent of the data communication protocol used to transport the messages between the client and server computers. Open Financial Exchange 1.0 will use HTTP.

1. Open Financial Exchange at a Glance

The design of Open Financial Exchange is as a client and server system. An end-user uses a client application to communicate with a server at a financial institution. The form of communication is requests from the client to the server and responses from the server back to the client.

Open Financial Exchange uses the Internet Protocol (IP) suite to provide the communication channel between a client and a server. IP protocols are the foundation of the public Internet and a private network can also use them.

1. Data Transport

Clients use the HyperText Transport Protocol (HTTP) to communicate to an Open Financial Exchange server. The World Wide Web throughout uses the same HTTP protocol. In principle, a financial institution can use any off-the-shelf web server to implement its support for Open Financial Exchange.

To communicate by means of Open Financial Exchange over the Internet, the client must establish an Internet connection. This connection can be a dial-up Point-to-Point Protocol (PPP) connection to an Internet Service Provider (ISP) or a connection over a local area network that has a gateway to the Internet.

Clients use the HTTP POST command to send a request to the previously acquired Uniform Resource Locator (URL) for the desired financial institution. The URL presumably identifies a Common Gateway Interface (CGI) or other process on an FI server that can accept Open Financial Exchange requests and produce a response.

The POST identifies the data as being of type application/x-ofx. Use application/x-ofx as the return type as well. Fill in other fields per the HTTP 1.0 spec. Here is a typical request:

```
POST http://www.fi.com/ofx.cgi HTTP/1.0
User-Agent:MyApp 5.0
Content-Type: application/x-ofx
Content-Length: 1032
```

```
OFXHEADER:100
DATA:OFXSGML
VERSION:100
SECURITY:1
ENCODING:USASCII

<OFX>
... Open Financial Exchange requests ...
</OFX>
```

A blank line defines the separation between the HTTP headers and the start of the actual Open Financial Exchange data. A blank line also separates the Open Financial Exchange headers and the actual response. (See Chapter 2, for more information.)

The structure of a response is similar to the request, with the first line containing the standard HTTP result, as shown next. The content length is given in bytes.

```
HTTP 1.0 200 OK
Content-Type: application/x-ofx
Content-Length: 8732

OFXHEADER:100
DATA:OFXSGML
VERSION:100
SECURITY:1
ENCODING:USASCII

<OFX>
... Open Financial Exchange responses ...
</OFX>
```

1. Request and Response Model

The basis for Open Financial Exchange is the request and response model. One or more requests can be batched in a single file. This file typically includes a signon request and one or more service-specific requests. An FI server will process all of the requests and return a single response file. This batch model lends itself to Internet transport as well as other off-line transports. Both requests and responses are plain text files, formatted using a grammar based on Standard Generalized Markup Language (SGML). Open Financial Exchange is syntactically similar to HyperText Markup Language (HTML), featuring tags to identify and delimit the data. The use of a tagged data format allows Open Financial Exchange to evolve over time while continuing to support older clients and servers.

Here is a simplified example of an Open Financial Exchange request file. (This example does not show the Open Financial Exchange headers and the indentation is only for readability.) For complete details, see the more complete examples throughout this specification.

<OFX> <!-- Begin request data --> <SIGNONMSGSRQV1> <SONRQ> <!-- Begin signon --> <DTCLIENT>19961029101000 <!-- Oct. 29, 1996, 10:10:00 am --> <USERID>123-45-6789 <!-- User ID (that is, SSN) --> <USERPASS>MyPassword <!-- Password (SSL encrypts whole) --> <LANGUAGE>ENG <!-- Language used for text --> <FI> <!-- ID of receiving institution --> <ORG>NCH <!-- Name of ID owner -->

```
<FID>1001 <!-- Actual ID --> </FI> <APPID>MyApp <APPVER>0500
</SONRQ> <!-- End of signon --> </SIGNONMSGSRQV1>
<BANKMSGSRQV1> <STMTTRNRQ> <!-- First request in file -->
<TRNUID>1001 <STMTRQ> <!-- Begin statement request -->
<BANKACCTFROM> <!-- Identify the account -->
<BANKID>121099999 <!-- Routing transit or other FI ID -->
<ACCTID>999988 <!-- Account number --> <ACCTTYPE>CHECKING
<!-- Account type --> </BANKACCTFROM> <!-- End of account ID -->
<INCTRAN> <!-- Begin include transaction --> <INCLUDE>Y <!--
Include transactions --> </INCTRAN> <!-- End of include transaction --
> </STMTRQ> <!-- End of statement request --> </STMTTRNRQ> <!--
End of first request --> </BANKMSGSRQV1></OFX> <!-- End of
request data -->
```
The response format follows a similar structure. Although a response such as a statement response contains all of the details of each transaction, each element is identified using tags.

The key rule of Open Financial Exchange syntax is that each tag is either an element or an aggregate. Data follows its element tag. An aggregate tag begins a compound tag sequence, which must end with a matching tag; for example, <AGGREGATE> ... </AGGREGATE>.

The actual file Open Financial Exchange sends is without any extra white space between tags.

1. Conventions

The conventions used in the detailed descriptions include:

- Required tags are in **bold**. Regular face indicates tags that are optional. Required means that a client will always include a tag in a request, and a server must always include a tag in a response.
- *Italic* shows a required or optional aggregate from a set of possible aggregates.
- Required tags occur once unless noted as one or more in the description, in which case the specification allows multiple occurrences.
- Optional tags occur once if present unless noted as zero or more in the description, in which case the specification allows multiple occurrences.
- Allowable specific values are listed, where applicable.
- A-*n* or N-*n*, specify those values that take general alphanumeric or pure numeric type values, where *n* indicates the maximum size.
- References to certain common value types, such as a dollar amount, are by name. Chapter 3 lists value types that can be referenced by name.

<REQUIREDTAG>	Required tag (1 or more)
<REQUIREDTAG2>	Required tag that occurs only once
<OPTIONALTAG>	Optional tag; this particular one can occur multiple times (0 or more)
<SPECIFIC>	Values are A, B, and C
<ALPHAVALUE>	Takes an alphanumeric value up to 32 characters, *A-32*

1. Structure

This chapter describes the basic structure of an Open Financial Exchange request and response. Structure includes headers, basic syntax, and the Signon request and response. This chapter also describes how Open Financial Exchange encodes external data, such as bit maps.

Open Financial Exchange data consists of some headers plus one or more Open Financial Exchange data blocks. Each block consists of a signon message and zero or more additional messages. When sent over the internet using HTTP, standard HTTP and multi-part MIME headers and formats surround the Open Financial Exchange data. A simple file that contained only Open Financial Exchange data would have the following form:

```
HTTP headers
MIME type application/x-ofx
Open Financial Exchange headers
Open Financial Exchange SGML block 1
```

A more complex file that contained multiple Open Financial Exchange data blocks and additional Open Financial Exchange data would have this form:

```
HTTP headers
MIME type multipart/x-mixed-replace; boundary =--boundary-
---boundary---
MIME type application/x-ofx
        Open Financial Exchange headers
        Open Financial Exchange SGML block 1
        Open Financial Exchange SGML block 2
---boundary---
        MIME type image/jpeg
                FI logo
```

1. HTTP Headers

Data delivered by way of HTTP places the standard HTTP result code on the first line. HTTP defines a number of status codes. Servers can return any standard HTTP result. However, FIs should expect clients to collapse these codes into the following three cases:

200	OK	The request was processed and a valid Open Financial Exchange result is returned.

400s	Bad request	The request was invalid and was not processed. Clients will report an internal error to the user.
500s	Server error	The server is unavailable. Clients should advise the user to retry shortly.

NOTE: *Open Financial Exchange returns a code 400 only if it cannot parse the file. Open Financial Exchange handles content errors such as wrong PIN, or invalid account, by returning a valid Open Financial Exchange response along with code 200.*

Open Financial Exchange requires the following HTTP standard headers:

Content-type	application/x-ofx	The MIME type for Open Financial Exchange
Content-length	length	Length of the data after removing HTTP headers

When responding with multi-part MIME, the main type will be multi-part/x-mixed-replace; one of the parts will use application/x-ofx.

1. Open Financial Exchange Headers

The intent of Open Financial Exchange is for use with a variety of transports and to provide sufficient version control capabilities for future expansion. To support this goal, the contents of an Open Financial Exchange file consist of a simple set of headers followed by contents defined by that header. "File format" means the entire content after removal of any transport headers. The HTTP transport described in this document, means without the HTTP and MIME headers.

The Open Financial Exchange headers are in a simple *tag:value* syntax and terminated by a blank line. Open Financial Exchange always sends headers unencrypted, even if there is application-level encryption in use for the remaining contents. The first entry will always be OFXHEADER with a version number. This entry will help identify the contents as an Open Financial Exchange file, and provides the version of the Open Financial Exchange headers that follow (not of the content itself). For example:

```
OFXHEADER:100
```

This document defines version 1.0 of the headers to contain at least the following additional tags:

```
DATA:OFXSGML
VERSION:100
SECURITY:
ENCODING:
```

```
CHARSET:
COMPRESSION:
OLDFILEUID:
NEWFILEUID:
```

The data tag identifies the contents as being in OFX SGML form. VERSION identifies the version type as OFXSGML data. In the case of OFXSGML, it translates to the version of the Document Type Definition (DTD) that it uses for parsing. The ENCODING and CHARSET tags define the interpretation of the character data. See Chapter 5, "International Support" for more information on these tags. Chapter 4 describes the security tag. A future version of this specification will define compression.

Open Financial Exchange uses OLDFILEUID and NEWFILEUID to support error recovery. They are not present when clients are not requesting error recovery. (See Chapter 6, "Data Synchronization")

A blank line follows the last tag. Then (for type OFXSGML), the SGML-readable data begins with the <OFX> tag.

NOTE: *Here, VERSION provides the overall version of the DTD. The <OFX> block describes the specific message set versions used, shown later in this chapter.*

1. The Meaning of Version Numbers

The OFXHEADER value should only change its major number if an existing client is unable to process the new header. This can occur because of a complete syntax change in a header, or a significant change in the semantics of an existing tag-not the entire response. You can add new tags as long as clients can function without understanding them.

You should add new values for a data tag only when you introduce an entirely new syntax. In the case of OFXSGML, a new syntax would have to be non-SGML compliant to warrant a new data value. It is possible that there will be more than one syntax in use at the same time to meet different needs.

The intent of the header version tag is to identify syntactic changes. In the case of OFXSGML, this corresponds to the DTD. Purely for identification purposes, each change will increment the minor number of the version tag. If you introduce an incompatible change so that an older DTD can not parse the file, the major number will change. See the general discussion of message sets and version control, later in this chapter.

1. SGML Details

2. Compliance

SGML is the basis for Open Financial Exchange. There is a DTD that formally defines the SGML wire format. However, Open Financial Exchange is not completely SGML-*compliant* because the specification allows unrecognized tags to be present. It requires clients and servers to skip over the unrecognized material. That is, if <XYZ>qqq</XYZ> appeared and a client or server cannot recognize <XYZ>, the server should ignore that tag and its enclosed data. A fully-compliant SGML parser would not *validate* an Open Financial Exchange document if it contained any tags that the DTD does not define.

Although SGML is the basis for the specification, and the specification is largely compliant with SGML, do not assume Open Financial Exchange supports any SGML features not documented in this specification. The intent is to allow parsing to be as simple as possible, while retaining compatibility with the SGML world.

1. Special Characters

The following characters are special to SGML. Use the given alternative sequence to represent them:

< (less than)	&lt;
> (greater than)	&gt;
& (ampersand)	&amp;

For example, the string "AT&amp;T" encodes "AT&T."

A special case applies in specific tags that can accept HTML-formatted strings, such as e-mail records. These accept SGML marked section syntax to hide the HTML from the Open Financial Exchange parser. You must prefix strings with "<![CDATA ["and suffixed with"]]>." Within these bounds, treat the above characters literally without an escape. See the Chapter 9 for an example.

1. Open Financial Exchange SGML Structure

2. Overview

Open Financial Exchange hierarchically organizes request and response blocks:

Top Level <OFX>
Message Set and Version <*XXX*MSGSVn>
Synchronization Wrappers <YYYSYNCRQ>, <YYYSYNCRS>
Transaction Wrappers <YYYTRNRQ>, <YYYTRNRS>
Specific requests and responses

The following sections describe each of these levels.

1. Top Level

An Open Financial Exchange request or response has the following top-level form:

<**OFX**>	Opening tag
... Open Financial Exchange requests or responses ...	0 or more transaction requests and responses inside appropriate message set aggregates

</OFX>	Closing tag for the Open Financial Exchange record

This chapter specifies the order of requests and responses.

A single file can contain multiple <OFX> ... </OFX> blocks. A typical use of multiple blocks is to request in a single file information associated with different users.

1. Messages

A message is the unit of work in Open Financial Exchange. It refers to a request and response pair, and the status codes associated with that response. For example, the message to download a bank statement consists of the request <STMTRQ> and the response <STMTRS>. In addition, with the exception of the signon message, each message includes a *transaction wrapper*. These aggregates add a transaction unique ID <TRNUID>, and for responses, a <STATUS> aggregate, to the basic request and response.

For messages subject to synchronization (see Chapter 6), a third layer of aggregates is also part of a message definition: a synchronization request and response. These add a token and, in some cases, other information to the transactions.

Open Financial Exchange uses the following naming where the *XXX* message includes:

- Basic request <*XXX***RQ**> and response <*XXX***RS**>
- Transaction wrapper <*XXX***TRNRQ**> and <*XXX***TRNRS**>
- If needed, synchronization wrapper <*XXX***SYNCRQ**> and <*XXX***SYNCRS**>

In a few cases, a small number of related basic requests and responses share a transaction and synchronization wrapper. The term message will still apply to each request and response; only the naming scheme will not hold in those cases.

1. Message Sets and Version Control

Message sets are collections of messages. Generally they form all or part of what a user would consider a *service*, something for which they might have signed up, such as "banking." Message sets are the basis of version control, routing, and security. They are also the basis for the required ordering in Open Financial Exchange files.

Within an Open Financial Exchange block, Open Financial Exchange organizes messages by message set. A message set can appear at most once within an Open Financial Exchange block. All messages from a message set must be from the same version of that message set.

For each message set of *XXX* and version *n*, there exists an aggregate named <*XXX*MSGSV*n*>. (Compare with <*XXX*MSGSETV*n*> in Chapter 7.) All of the messages from that message set must be inside the appropriate message set aggregate. In the following example, the Open Financial Exchange block contains a signon request inside the signon message set, and two statement requests and a transfer request inside the bank message set.

```
<OFX>
        <SIGNONMSGSRQV1>        <!-- Signon message set -->
                <SONRQ>                         <!-- Signon message -->
                ...
                </SONRQ>
        </SIGNONMSGSRQV1>

        <BANKMSGSRQV1>          <!-- Banking message set -->
                <STMTTRNRQ>             <!-- Statement request -->
                ...
                </STMTTRNRQ>
                <STMTTRNRQ>             <!-- Another stmt request -->
                ...
                </STMTTRNRQ>
                <INTRATRNRQ>            <!-- Intra-bank transfer request -->
                ...
                </INTRATRNRQ>
        </BANKMSGSRQV1>
</OFX>
```

Message sets, if used at all, must appear in the following order:

- Signon
- Signup
- Banking
- Credit card statements
- Investment statements
- Interbank funds transfers
- Wire funds transfers
- Payments
- General e-mail
- Investment security list
- FI Profile

The definition of each message set can further prescribe an order of its messages within that message set.

1. Transactions

Other than the signon message, each request is made as a transaction. Transactions contain a client-assigned globally unique ID, optional client-supplied pass-back data, and then the record for the specific request. A transaction similarly wraps each response. The response transaction returns the client ID sent in the request, along with a status message, the pass-back data if present, and the specific response record. This technique allows a client to track responses against requests.

The <STATUS> aggregate, defined in Chapter 3, provides feedback on the processing of the request. If the <SEVERITY> of the status is ERROR, the server provides no specific response record. Otherwise, the response will be complete even though some warning might have occurred.

Clients can send additional information in <CLTCOOKIE> that servers will return in the response. This allows clients that do not maintain state, and thus do not save TRNUIDs, to cause some additional descriptive information to be present in the response. For example, a client might identify a request as relating to a user or a spouse.

In some countries some transactions require a customer-supplied authorization number for each transaction. In those countries, the <TAN> element provides the means to pass this information to servers. As Open Financial Exchange is implemented in each country, the specification will define the specific requirements for the use of <TAN> in each country.

A typical request is as follows:

<*XXX*TRNRQ>	Transaction-request aggregate
<TRNUID>	Client-assigned globally unique ID for this transaction *trnuid*
<CLTCOOKIE>	Data to be echoed in the transaction response *A-32*
<TAN>	Transaction authorization number; used in some countries with some types of transactions. Country-specific documentation will define messages that require a TAN, *A-80*
specific request	Aggregate for the specific request
</*XXX*TRNRQ>	

A typical response is as follows:

<*XXX*TRNRS>	Transaction-response aggregate
<TRNUID>	Client-assigned globally unique ID for this transaction, *trnuid*
<CLTCOOKIE>	Client-provided data, **REQUIRED** if provided in request, *A-32*
<STATUS>	Status aggregate
</STATUS>	
response record	Aggregate for the specific response
</*XXX*TRNRS>	

1. The Signon Message Set

The Signon message set includes the signon message and the PIN change message, and must appear in that order. The <SIGNONMSGSRQV1> and <SIGNONMSGSRSV1> aggregates wrap the message.

1. Signon <SONRQ> <SONRS>

The signon record identifies and authenticates a user to an FI. It also includes information about the application making the request, because some services might be appropriate only for certain clients. Every Open Financial Exchange request contains exactly one <SONRQ>. Every response must contain exactly one <SONRS> record.

Use of Open Financial Exchange presumes that FIs authenticate each customer and then give the customer access to one or more accounts or services. If passwords are specific to individual services or accounts, a separate Open Financial Exchange request will be made for each distinct user ID or password required. This will not necessarily be in a manner visible to the user. Note that some situations, such as joint accounts or business accounts, will have multiple user IDs and multiple passwords that can access the same account.

FIs assign user IDs for the customer. It can be the customer's social security number, but the client will not make any assumptions about the syntax of the ID, add check-digits, or do similar processing.

To improve server efficiency in handling a series of Open Financial Exchange request files sent over a short period of time, clients can request that a server return a <USERKEY> in the signon response. If the server provide a user key, clients will send the <USERKEY> in instead of the user ID and password in subsequent sessions, until the <USERKEY> expires. This allows servers to authenticate subsequent requests more quickly.

The client returns <SESSCOOKIE> if it sent one in a previous <SONRS>. Servers can use this value to track client usage but cannot assume that all requests come from a single client, nor can they deny service if they did not expect the returned cookie. Use of a backup file, for example, would lead to an unexpected <SESSCOOKIE> value that should nevertheless not stop a user from connecting.

Servers can request that a consumer change his or her password by returning status code 15000. Servers should keep in mind that only one status code can be returned. If the current signon response status should be 15500 (invalid ID or password), the request to change password will need to wait until an otherwise successful signon is achieved.

1. Record Request <SONRQ>

<SONRQ>	Record- request aggregate
<**DTCLIENT**>	Date and time of the request from the client computer, *datetime*
<USERID>	User identification string. Use <USERID> & <USERPASS>, or <USERKEY>, but not both; *A-32*
<USERPASS>	User password on server - either <USERID> & <USERPASS> are used, or <USERKEY>, but not both; *A-32*
<USERKEY>	Login using previously authenticated context - use <USERID> & <USERPASS>, or <USERKEY>, but not

	both; *A-64*
<GENUSERKEY>	Request server to return a USERKEY for future use, *Boolean*
<LANGUAGE>	Requested language for text responses, *language*
<SESSCOOKIE>	Session cookie, value received in previous <SONRS>, not sent if first login or if none sent by FI *A-1000*
<FI>	Financial-Institution-identification aggregate
</FI>	
<APPID>	ID of client application, *A-5*
<APPVER>	Version of client application, *N-4* (6.00 encoded as 0600)
</SONRQ>	

1. Response <SONRS>

<SONRS>	Record-response aggregate
<STATUS>	Status aggregate, see list of possible code values
<DTSERVER>	Date and time of the server response, *datetime*
<USERKEY>	Use user key that instead of USERID and USERPASS for subsequent requests. TSKEYEXPIRE can limit lifetime
<TSKEYEXPIRE>	Date and time that USERKEY expires
<LANGUAGE>	Language used in text responses, *language*
<DTPROFUP>	Date and time of last update to profile information for any service supported by this FI (see Chapter 7), *datetime*
<DTACCTUP>	Date and time of last update to account information (see Chapter 8), *datetime*
<FI>	Financial-Institution-identification aggregate
</FI>	
<SESSCOOKIE>	Session cookie that the client should return on the next <SONRQ> *A-1000*
</SONRS>	

List of status code values for the <CODE> element of <STATUS>:

0	Success (INFO)

2000	General error (ERROR)
15000	Must change PIN (INFO)
15500	Signon (for example, user ID or password) invalid (ERROR)
15501	Customer account already in use (ERROR)
15502	PIN Lockout (ERROR)

1. Financial Institution ID <FI>

Some service providers support multiple FIs, and assign each FI an ID. The signon allows clients to pass this information along, so that providers will know to which FI the user is actually doing a signon.

<FI>	FI-record aggregate
<ORG>	Organization defining this FI name space, *A-32*
<FID>	Financial Institution ID (unique within <ORG>), *A-32*
</FI>	

1. PIN Change <PINCHRQ> <PINCHRS>

The signon sends a request to change a customer password as a separate request. The transaction request <PINCHTRNRQ> aggregate contains <PINCHRQ>. Responses are also inside transaction responses <PINCHTRNRS>.

Password changes pose a special problem for error recovery. If the client does not receive a response, it does not know whether the password change was successful or not. Open Financial Exchange recommends that servers accept either the old password or the new password on the connection following the one containing a password change. The password used becomes the new password.

<PINCHRQ>	PIN-change-request aggregate
<USERID>	User identification string. Often a social security number, but if so, does not include any check digits, *A-32*
<NEWUSERPASS>	New user password, *A-32*
</PINCHRQ>	

<PINCHRS>	PIN-change-response aggregate
<USERID>	User identification string. Often a social security number, but if so, does not include any check digits, *A-32*
<DTCHANGED>	Date and time the password was changed, *datetime*

</PINCHRS>	

1. Status Code Values for the <CODE> Element of <STATUS>

0	Success (INFO)
2000	General error (ERROR)
15503	Could not change PIN (ERROR)

1. Examples

User requests a password change:

```
<PINCHTRNRQ>
        <TRNUID>888
        <PINCHRQ>
                <USERID>123456789
                <NEWUSERPASS>5321
        </PINCHRQ>
</PINCHTRNRQ>
```

The server responds with:

```
<PINCHTRNRS>
        <TRNUID>888
        <STATUS>
                <CODE>0
                <SEVERITY>INFO
        </STATUS>
        <PINCHRS>
                <USERID>123456789
        </PINCHRS>
</PINCHTRNRS>
```

1. External Data Support

Some data, such as binary data, cannot be easily sent directly within SGML. For these situations, the specification will define a tag that contains a reference to some external data. The way that clients pick up the external data depends on the transport used. For the HTTP-based transport described in this document, servers can send the data in one of two ways:

- Send the same response, using multi-part MIME types to separate the response into basic
  Open Financial Exchange and one or more external data files
- Client can make a separate HTTP get against the supplied URL, if it really needs the data

For example, to retrieve a logo, a <GETMIMERS> might answer a <GETMIMERQ> as follows:

```
<GETMIMERS>
        <URL>https://www.fi.com/xxx/yyy/zzz.html
</GETMIMERS>
```

If the file sent includes the same response using multi-part MIME, clients will assume it has the local file, zzz.jpg.

1. Extensions to Open Financial Exchange

An organization that provides a customized client and server that communicate by means of
Open Financial Exchange might wish to add new requests and responses or even specific elements to existing requests and responses. To ensure that each organization can extend the specification without the risk of conflict, Open Financial Exchange defines a style of tag naming that lets each organization have its own name space.

Organizations can register a specific tag name prefix. (The specific procedure or organization to manage this registration will be detailed at a later time.) If an organization registers "ABC," then they can safely add new tags named <ABC.SOMETHING> without

- Colliding with another party wishing to extend the specification
- Confusing a client or server that does not support the extension

The extensions are not considered proprietary. An organization is free to publish their extensions and encourage client and server implementers to support them.

All tag names that do not contain a period (.) are reserved for use in future versions of the core
Open Financial Exchange specification.

1. Common Aggregates, Elements, and Data Types

2. Common Aggregates

This section describes aggregates used in more than one service of Open Financial Exchange (for example, investments and payments).

1. Identifying Financial Institutions and Accounts

Open Financial Exchange does not provide a universal space for identifying financial institutions, accounts, or types of accounts. The way to identify an FI and an account at that FI depends on the service. For information about service-specific ID aggregates, see Chapters 11, 12, and 13 on banking, payments, and investments.

1. Balance Records <BAL>

Several responses allow FIs to send an arbitrary set of balance information as part of a response, for example a bank statement download. FIs might want to send information on outstanding balances, payment dates, interest rates, and so forth. Balances can report the date the given balance reflects in <DTASOF>.

<BAL>	Balance-response aggregate
<NAME>	Balance name, *A-20*
<DESC>	Balance description, *A-80*

<BALTYPE>	Balance type.
	DOLLAR = dollar (value formatted DDDDcc)
	PERCENT = percentage (value formatted XXXX.YYYY)
	NUMBER = number (value formatted as is)

<VALUE>	Balance value. Interpretation depends on <BALTYPE> field, *N-20*
<CURRENCY>	If dollar formatting, can optionally include currency
<DTASOF>	Effective date of the given balance, *datetime*
</BAL>	

1. Error Reporting <STATUS>

To provide as much feedback as possible to clients and their users, Open Financial Exchange defines a <STATUS> aggregate. The most important element is the code that identifies the error. Each response defines the codes it uses. Codes 0 through 2999 have common meanings in all Open Financial Exchange transactions. Codes from 3000 and up have meanings specific to each transaction.

<STATUS>	Error-reporting aggregate.
<CODE>	Error code, *N-6*
<SEVERITY>	Severity of the error: INFO = Informational only WARN = Some problem with the request occurred but valid response still present ERROR = A problem severe enough that response could not be made
<MESSAGE>	A textual explanation from the FI. Note that clients will generally have messages of their own for each error ID. Use this tag only to provide more details or for the General errors.
</STATUS>	

stadyn_image6

stadyn_image7

For general errors, the server can respond with one of the following <CODE> values. However, not all codes are possible in a specific context.

0	Success (INFO)
2000	General error (ERROR)
2021	Unsupported version (ERROR)

NOTE: *Clients will generally have error messages based on <CODE>. Therefore, do not use <MESSAGE> to replace that text. Use <MESSAGE> only to explain an error not well described by one of the defined CODEs, or to provide some additional information.*

1. Common Elements

This section defines elements used in several services of Open Financial Exchange. The format of the value is either alphanumeric (A-*n*) or numeric (N-*n*) with a maximum length *n*; or as a named type. Section 3.3 describes the named types.

1. Financial Institution Transaction ID <FITID>

Format: *A-255*

An FI assigns an <FITID> to uniquely identify a transaction. Its primary purpose is to allow a client to detect duplicate responses. Open Financial Exchange intends <FITID> for use in statement download applications, where every transaction requires a unique ID; not just those that are client-originated or server-originated.

FITIDs must be unique within the scope of the requested transactions (that is, within an account) but need not be sequential or even increasing. Clients should be aware that FITIDs are not unique across FIs. If a client performs the same type of request within the same scope at two different FIs, clients will need to use FI + account + <FITID> as a unique key in a client database.

Usage: Bank statement download, investment statement download

1. Server-Assigned ID <SRVRTID>

Format: *A-10*

A <SRVRTID> is a server-assigned ID. It should remain constant throughout the lifetime of the object on the server. The client will consider the SRVRTID as its "receipt" or confirmation and will use this ID in any subsequent requests to change, delete, or inquire about this object.

Where the context allows, it is possible for a server to use the same *value* for a given server object for both <SRVRTID> and <FITID>, but the client will not know this. SRVRTIDs need be unique only within the scope of the requests and responses they apply to, such as an account number. Like <FITID>, a <SRVRTID> is not unique across FIs and clients might need to use FI + <SRVRTID> if a client requires a unique key.

Usage: Payments, Banking

1. Client-Assigned Transaction UID <TRNUID>

Format: *A-36*

Open Financial Exchange uses TRNUIDs to identify transactions, specifically <*XXX*TRNRQ>. Clients expect a server to return the same <TRNUID> in the corresponding response and can use this to match up requests and responses. Servers can use TRNUIDs to reject duplicate requests. Because multiple clients might be generating requests to the same server, transaction IDs need to be unique across clients. Thus, <TRNUID> must be a globally unique ID.

The Open Software Foundation Distributed Computing Environment standards specify a 36-character hexadecimal encoding of a 128-bit number and an algorithm to generate it. Clients are free to use their own algorithm, to use smaller TRNUIDs, or to relax the uniqueness requirements if in their particular application it makes sense. However, it is **RECOMMENDED** that clients allow for the full 36 characters in responses to work better with other clients.

Usage: All services

1. Token <TOKEN>

Format: *A-10*

Open Financial Exchange uses **<**TOKEN> as part of data synchronization requests to identify the point in history that the client has already received data, and in responses to identify the server's current end of history. See Chapter 6, "Data Synchronization," for more information.

<TOKEN> is unique within an FI and the scope of the synchronization request. For example, if the synchronization request includes an account ID, the <TOKEN> needs be unique only within an account. Servers are free to use a <TOKEN> that is unique across the entire FI. Clients must save separate <TOKEN>s for each account, FI, and type of synchronization request.

Usage: All synchronization requests and responses

1. Transaction Amount <TRNAMT>

Format: *Amount*

Open Financial Exchange uses <TRNAMT> in any request or response that reports the total amount of an individual transaction.

Usage: Bank statement download, investment statement download, payments

1. Memo <MEMO>

Format: *A-255*

A <MEMO> provides additional information about a transaction.

Usage: Bank statement download, investment statement download, payments, transfers

1. Date Start and Date End <DTSTART> <DTEND>

Format: *Datetime*

Open Financial Exchange uses these tags in requests to provide guidance to the FI about the range of response that is desired. It also uses them in responses to let clients know what the FI was actually able to produce.

In requests, the following rules apply:

- If <DTSTART> is absent, the client is requesting all available history (up to the <DTEND>, if specified). Otherwise, it indicates the *inclusive* date and time in history where the client expects servers to start sending information.
- If <DTEND> is absent, the client is requesting all available history (starting from <DTSTART>, if specified). Otherwise, it indicates the *exclusive* date and time in history where the client expects servers to stop sending information.

In responses, the following rules apply:

- <DTSTART> is the date and time where the server began *looking* for information, not necessarily the date of the earliest returned information. If the response <DTSTART> is later than the requested <DTSTART>, clients can infer that the user has not signed on frequently enough to ensure that the client has retrieved all information. If the user has been calling frequently enough, <DTSTART> in the response will match <DTSTART> in the request.
- <DTEND> is the date and time that, if used by the client as the next requested <DTSTART>, it would pick up exactly where the current response left off. It is the *exclusive* date and time in history where the server stopped *looking* for information, based on the request <DTEND> rules.

In all cases, servers are **REQUIRED** to use a "system add datetime" as the basis for deciding which details match the requested date range. For example, if an FI posts a transaction dated Jan 3 to a user's account on Jan 5, and a client connects on Jan 4 and again on Jan 6, the server is **REQUIRED** to return that Jan 3 dated transaction to the client when it calls on Jan 6.

Usage: Bank statement download, investment statement download

1. Common data types

2. Dates and Times

3. Basic Format

There is one format for representing dates, times, and time zones. The complete form is:

YYYYMMDDHHMMSS.XXX[*gmt offset*:*tz name*]

For example, "19961005132200.1234[-5:EST]" represents October 5, 1996, at 1:22 and 124 milliseconds p.m., in Eastern Standard Time. This is the same as 6:22 p.m. Greenwich Mean Time (GMT).

Tags specified as type *date* and generally starting with the letters "DT" will accept a fully formatted date-time-timezone as specified above. They will also accept values with fields omitted from the right. They assume the following defaults if a field is missing:

- YYYYMMDD: 12:00 midnight (the start of the day), GMT
- YYYYMMDDHHMMSS: GMT
- YYYYMMDDHHMMSS.XXX: GMT
- YYYYMMDDHHMMSS.XXX[-0500:EST]: Fully qualified

Open Financial Exchange identifies elements that require a time as having type *timestamp* and their tag name will start with the prefix TS. The timezone and milliseconds are still optional, and will default to GMT.

Take care when specifying an ending date without a time. If the last transaction returned for a bank statement download was Jan 5 1996 10:46 a.m. and if the <DTEND> was given as just Jan 5, the transactions on Jan 5 would be resent. If results are only available daily, then just using dates and not times will work correctly.

NOTE: *Open Financial Exchange does not require servers or clients to use the full precision specified. However, they are* ***REQUIRED*** *to accept any of these forms without complaint.*

Some services extend the general notion of a *date* by adding special values, such as "TODAY." These special values are called "smart dates." Specific requests indicate when to use these extra values, and list the tag as having a special data type.

1. Time Zone Issues

Several issues arise when a customer and the FI are not in the same time zone, or when a customer moves a computer into new time zones. In addition, it is generally unsafe to assume that computer users have correctly set their time or timezone.

Although most transactions are not sensitive to the exact time, they often are sensitive to the date. In some cases, time zone errors lead to actions occurring on a different date than intended by the customer. For this reason, servers should always use a complete local time plus GMT offset in any datetime values in a response. If a customer's request is for 5 p.m. EST, and a server in Europe responds with 1 a.m. MET the next day, a smart client can choose to warn the customer about the date shift.

Clients that maintain local state, especially of long-lived server objects, should be careful how they store datetime values. If a customer initiates a repeating transaction for 5 p.m. EST, then moves to a new time zone, the customer might have intended that the transaction remain 5 p.m. in the new local time, requiring a change request to be sent to the server. If, however, they intended it to remain fixed in server time, this would require a change in the local time stored in the client.

1. Amounts, Prices, and Quantities

2. Positive and Negative Signs

Unless otherwise noted in the specification, Open Financial Exchange always signs amounts and quantities from the perspective of the customer. Some typically negative amounts:

- Investment buy amount, investment sell quantity
- Bank statement debit amounts, checks, fees
- Credit card purchases
- Margin balance (unless the FI owes the client money)

Some typically positive amounts:

- Investment sell amount, investment buy quantity
- Bank statement credits
- Credit card payments
- Ledger balance (unless the account is overdrawn)

Amount: All amount-valued tags are sent with a decimal point or comma, as in "XXXX.XX." There should not be any punctuation separating thousands, millions, and so forth. The maximum value accepted depends on the client.

Quantity: Use decimal notation.

Unitprice: Use decimal notation. Unless specifically noted, prices should always be positive.

Rate: Use decimal notation, with the rate specified out of 100%. For example, 5.2 is 5.2%.

Some services define special values, such as INFLATION, which you can use instead of a designated value. Open Financial Exchange refers to these as "smart types," and identifies them in the specification.

1. Language

Open Financial Exchange identifies human-readable language-for such things as status messages and e-mail-with a three-letter code based on ISO-639.

1. Basic data types

Boolean: Y = yes or true, N = no or false.

URL: String form of a World Wide Web Uniform Resource Location. It should be fully qualified including protocol, host, and path.

1. Security

2. Security Solutions

Open Financial Exchange carries financial information over the Internet in such a way to provide privacy, message integrity, and authentication for applications at the appropriate level. Each service within Open Financial Exchange requires a certain level of security. Online banking and payments require strong secrecy, whereas stock quotations consist of publicly available information and consequently have a much weaker secrecy requirement.

Some Internet protocols, such as HTTPS (which uses Secure Socket Layer version 3, SSLv3), provide channel-level security. When the security requirement exceeds that provided by the channel, you must use an application-level protocol.

To address these various needs, Open Financial Exchange allows a range of security solutions. Open Financial Exchange 1.0 supports online banking and payment functions for which strong channel security is currently appropriate. Future releases will support a wider array of services, some of which will require more elaborate trust models. Application-level protection will secure the latter.

Open Financial Exchange security properties include:

- SSL - protects information during transmission over the Internet between a customer and an FI
- Application layer security - encrypts and formats messages using RSA Data Security PKCS#7 techniques
- New cryptographic options and enhancements when available - will enhance Open Financial Exchange to provide these facilities

1. Determining Security Levels <OFXSEC> <TRANSPSEC>

Two elements in the FI profile, <OFXSEC> and <TRANSPSEC>, contain the security level a client should use to communicate with a server.

The valid values for <OFXSEC> are as follows:

NONE	No application level security
APPSEC	Use application level security

The <TRANSPSEC> element value is Boolean. If the value is YES, use channel-level security.

1. Channel-Level Security

Secure Socket Layer version 3 (SSLv3) provides channel-level security in Open Financial Exchange. SSLv3 provides confidentiality, message integrity, and implicit authentication. In Open Financial Exchange 1.0, channel-level security using SSLv3 is the primary form of security.

1. Security Requirements

Open Financial Exchange provides a method to exchange financial information over public networks. This necessitates strong security facilities and careful protocol design. The most commonly used facility, and trusted method for accomplishing many of these goals is SSL. The following sub-sections describe the most prominent requirements for security and how Secure Socket Layer (SSL) addresses these.

1. Privacy, Authentication, and Message Integrity

SSL provides a range of strong encryption methods for insuring confidentiality, and strong measures to insure that messages are not altered as they propagate over the Internet. User authentication is usually addressed at the application layer, not within SSL. Servers are configured with public key certificates that client application software verifies. This provides some measure of server authentication. Testing certificate revocation lists is not commonly performed. However, as these facilities emerge, client software will be written to support this need.

1. Facilities for Authorization

Open Financial Exchange messages typically provide user ID and password so that a service provider can authenticate the user. Once a system authenticates a user, the service provider must insure that the user is authorized to perform the requested actions. For example, the service provider must decide if the specified user is authorized to perform a transfer from the specified account. The service provider must also determine whether the user has exceeded allowed limits on withdrawals, whether the activity on this account is unusual given past history, and other context-sensitive issues.

1. Using SSL 3.0 in Open Financial Exchange

SSL version 3.0 provides a set of widely and commonly accepted methods for securing Internet transactions. These common methods within SSL are called CipherSuites. You can secure applications appropriately within SSL by specifying an ordered sequence of preferred CipherSuites (highest preference listed first). Servers select the strongest supported CipherSuite from the list provided by the client.

NOTE: *Passing username and password pairs in a weakly encrypted channel exposes this information to cryptographic attack. When implementing Open Financial Exchange, use the strongest available ciphers.*

You should not use the following CipherSuites because they are vulnerable to man-in-the-middle attacks during Open Financial Exchange message exchanges:

- SSL_DH_anon_EXPORT_WITH_RC4_40_MD5
- SSL_DH_anon_WITH_RC4_128_MD5
- SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA
- SSL_DH_anon_WITH_DES_CBC_SHA
- SSL_DH_anon_WITH_3DES_EDE_CBC_SHA

Setting tags to enable channel-level security in the FI profile advises the Open Financial Exchange application to use this security method. Usually, the service provider of the Web server configures the allowed CipherSuites within SSL.

1. Application-Level Security

While strong channel-level security is sufficient for the current suite of Open Financial Exchange transactions, there are features that channel security does not provide. These include (but are not limited to) data signing, non-repudiation, rational certificate management and revocation, and trust proxy. Where the trust model for an application requires such features to conduct the transaction safely, Open Financial Exchange stipulates the use of an application-level protocol. A future implementation guide will publish this protocol.

The standard method for providing application-level security is to rely upon the RSA Public Key Cryptography Standard (PKCS) message format. The PKCS #7 standard specifies a message format that is both cryptographically strong and flexible enough to provide sufficient facilities for evolution.

1. Requirements for Application-Layer Security

2. Privacy, Authentication, and Message Integrity

RSA Public Key Cryptography Standard #7 (PKCS#7) defines a rich set of message formats for securely exchanging information over public networks. These message formats provide for encrypting data using a combination of cryptographic techniques to leverage manageability of public key cryptography. It also utilizes the speed of block ciphers into a hybrid, which exploits the best properties of each.

PKCS#7 message encryption provides privacy. A digitally signed message (or applying HMAC) insures message integrity.

Use one of the following to define PKCS#7 messages: Data, Digitally Signed-Data, Enveloped-Data, or Digitally Signed and Enveloped-Data (also referred to as Sealed-Data). Open Financial Exchange can use Digested-Data, which digests application data before it embeds data within an Enveloped-Data object. However, it should not transmit this data over public networks without encryption applied.

1. Facilities for Authorization

As stated previously in the section 4.2, authentication and authorization are the responsibility of the service provider. Open Financial Exchange messages contain the information to enable authentication and authorization decisions. With application-level security that uses a digitally signed format, the verification of that signature provides an additional method of authenticating the user.

1. Using Application-level Encryption in Open Financial Exchange

Open Financial Exchange applications requiring a sophisticated trust model require more facilities than those provided by SSL. If an Open Financial Exchange application requires only point-to-point security, SSL version 3.0 provides adequate facilities for message security. However, if the application requires more directed, specific forms of security, then use the appropriate PKCS#7 message formats for the application. An example of this might be a stock trading application issuing orders whose values demand that the security level be high, and where Open Financial Exchange treats the message with special handling instructions.

Recommended cryptographic techniques for Open Financial Exchange application security are:

- RC4 for bulk encryption (using 40 bits for exportable applications, 128 for North America)
- RSA encryption of bulk encryption keys and digital signatures
- SHA-1 as a secure hash algorithm

In the absence of digital signatures, Open Financial Exchange applications should utilize the HMAC keyed MAC algorithm, using SHA-1 as a secure hash function.

When you set the tags for application-layer security-which determines whether to use PKCS#7 message format-in the FI profile, the application software uses these facilities.

1. International Support

2. Language 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, specific character set, and the specific language.

The outer Open Financial Exchange headers specify the encoding and character set, as described Chapter 2. Current encoding values are ASCII and UNICODE. For ASCII, character set values are code pages. Unicode ignores the character set *per se* although it still requires the syntax. 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 requested language, they must return an error and not process the rest of the transactions.

1. Currency <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 will 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> records to overcome this limitation, which do 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.

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

In some cases, Open Financial Exchange will define transaction responses so that amounts have been converted to the home currency. However, Open Financial Exchange will allow 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.

1. Country-Specific Tag Values

Some of the tags in Open Financial Exchange have values that are country-specific. For example, <USPRODUCTTYPE> is only useful 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 will 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, it is important that Open Financial Exchange defines 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.

1. Data Synchronization

2. Overview

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.

1. Background

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 will 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.

1. Data 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 it would have sent and a way to identify a position in the history. This ID could be a timestamp, or be based on its existing state information.

NOTE: *Open Financial Exchange does not require servers to store these 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.*

1. Data 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 will identify the specific 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 special data synchronization for this type of response.

The basis for synchronization is a *token* as defined by the <TOKEN> tag. The server is free to 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:

- Timestamp
- 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 will be subject to a maximum size; see Chapter 3, "Common Aggregates, Elements, and Data Types." Tokens need to be unique only with respect to a specific 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 a specific account.

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

NOTE: *A token is unrelated to a <TRNUID>, <SRVRTID>, or <FITID>. Each of these 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. Clients will send a specific synchronization request, including the current token for that request. The response will be a set of individual responses more recent than the given token, along with a new token.

- Synchronization with new requests - A client can request synchronization as part of the response to any new requests it has. It gives the old token. The response should include 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 client 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 particular 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>. 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.

1. Conflict 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 will always synchronize 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> option. 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.

1. Synchronization vs. Refresh

There are some situations, and some types of clients, where it is preferable for a client to ask a server to send everything it knows, rather than just receive a set of changes. For example, a situation where a client that has not connected often enough has lost synchronization. An example of "type" might be a completely stateless client, such as a web browser. This choice is made during client implementation. 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, you can 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 just wants 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:

<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*

NOTE: *Open Financial Exchange requires one each of* <TOKEN>, <TOKENONLY>, *or* <REFRESH>.

1. Typical 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.

1. Typical 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>.

1. Simultaneous 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, insure 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 for multiple users.

1. Synchronization 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 server both support full synchronization. This section first describes lite synchronization, and then explains the rules that clients and servers use to decide how to communicate.

1. Lite Synchronization

Lite synchronization requires servers to accept all synchronization messages, but does not require them to keep any history or tokens. Responses need only be sent 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 Chapter 3.

Servers use the following rules:

- If these tags are absent, the client is not requesting error recovery protection for this file. The server does not need to save a copy of the response.
- If the NEWFILEUID matches a file saved on the server, then the client is in error recovery. The server must ignore the new requests in this file and instead send back the matching saved response file.
- If the 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-datafile usage, it might need to save several files for a given user. A server should save 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.

1. Relating Synchronization and Error Recovery

Client and server developers should first decide whether they will support full synchronization or not. 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 severs 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.

1. Examples

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. See Chapter 102 for full details.

These simplified examples, show without the outer <OFX> layer, <SIGNON>, and so forth.Client A requests synchronization:

```
<PMTSYNCRQ>
        <TOKEN>123
        <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>
```

```
        <STATUS>
            ... status details
        </STATUS>
        <TRNUID>123
        <PMTRS>
            ... details on a payment response
        </PMTRS>
    </PMTTRNRS>
    <PMTTRNRS>
        <STATUS>
            ... status details
        </STATUS>
        <TRNUID>546
        <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
        <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.

1. FI Profile

2. Overview

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 timestamp 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.

1. Message 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.

1. Version 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. Considering the client capabilities, it exchanges 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. Its version 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.

1. Batching 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

1. Profile 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 it returns an error.

Profile requests are not subject to synchronization. Use the <PROFTRNRQ> transaction tag.

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

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 may have a different URL and/or signon realm.

The intent of the SERVICE option for client routing is to support 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.

1. Profile Response

The response includes message set descriptions, signon information, and general contact information.

<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
<ADDR2>	FI address, line 2
<ADDR3>	FI address, line 3
<CITY>	FI address city
<STATE>	FI address state
<POSTALCODE>	FI address postal code
<COUNTRY>	FI address country
<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-32*
<SYNCMODE>	FULL for full synchronization capability LITE for lite synchronization capability
<RESPFILEER>	Y if server supports response-file based error recovery, *Boolean*
</PROFRS>	

See the Chapter 6 for more information on <SYNCMODE> and <RESPFILEER>.

1. Message 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:

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

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

<*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:

<MSGSETCORE>	Common-message-set-information aggregate
<VER>	Version number, *N-5* (version 1.0 formatted as 100)
<URL>	URL where messages in this set are to be sent
<OFXSEC>	Security level required for this message set; see Chapter 4
<TRANSPSEC>	Y if transport security must be used, N if not used; *Boolean*
<SIGNONREALM>	Signon realm to use with this message set
<LANGUAGE>	One or more languages supported
</MSGSETCORE>	

1. Signon 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.

<SIGNONINFO>	Signon-information aggregate
<SIGNONREALM>	Identifies this realm
<MIN>	Minimum number of password characters
<MAX>	Max number of password characters
<ALPHA>	Y if alphabetic characters are allowed, *Boolean*
<NUMERIC>	Y if numeric characters are allowed, *Boolean*
<CASESEN>	Y if password is case-sensitive, *Boolean*
<SPECIAL>	Y if special characters are allowed, *Boolean*
<SPACES>	Y if spaces are allowed, *Boolean*
</SIGNONINFO>	

1. Status Codes

0	Success (INFO)
2000	General error (ERROR)

1. Profile Message Set Profile Information

stadyn_image8

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 that support profile information 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.

<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>	

1. Activation & Account Information

2. Overview

The Signup message set defines three messages to help users get setup with their FI:

- Enrollment - informs FI that a user wants to use Open Financial Exchange and requests that a password be returned
- Accounts - asks the FI to return a list of accounts, and the services supported for each account
- Activation - allows a client to tell the FI which services a user wants on each

account

There is also a message to request name and address changes.

Clients use the account information request on a regular basis to look for changes in a user's account information. A timestamp is part of the request so that a server has only to report new changes. Account activation requests are subject to data synchronization, and will allow multiple clients to learn how the other clients have been enabled.

In Open Financial Exchange files, the <SIGNUPMSGSV1> aggregate identifies the Signup message.

1. Approaches to User Sign-Up with Open Financial Exchange

The message sets in this chapter are designed to allow both FIs and clients to support a variety of sign-up procedures. There are four basic steps a user needs to go through to complete the sign-up:

1. **Select the FI**. Open Financial Exchange does not define this step or provide message sets to support it. Client developers and FIs can let a user browse or search this information on a web site, or might define additional message sets to do this within the client. At the conclusion of this step, the client will have some minimal profile information about the FI, including the set of services supported and the URL to use for the next step.
2. **Enrollment and password acquisition.** In this step, the user identifies and authenticates itself to the FI *without a password*. In return, the user obtains a password (possibly temporary) to use with Open Financial Exchange. FIs can perform this entire step over the telephone, through a combination of telephone requests and a mailed response, or at the FI web site. FIs can also use the Open Financial Exchange enrollment message to do this by means of the client. The response can contain a temporary password or users can wait for a mailed welcome letter containing the password.
3. **Account Information.** In this step, the user obtains a list of accounts available for use with Open Financial Exchange, and which specific services are available for each account. Even if users have enrolled over the telephone, clients will still use this message set to help users properly set up the accounts within the client. Clients periodically check back with the FI for updates.
4. **Service Activation.** The last step is to activate specific services on specific accounts. The activation messages support this step. Synchronization is applied to these messages to insure that other clients are aware of activated services.

The combination of media-interface through which an FI accomplishes these steps can vary. FIs might wish to do steps two through four over the telephone. Clients will still use

Open Financial Exchange messages in steps 3 and 4 to automatically set up the client based on the choices made by the user over the phone. Other FIs might wish to have the entire user experience occur within the client. Either way, the Open Financial Exchange sign-up messages support the process.

1. Users and Accounts

To support the widest possible set of FIs, Open Financial Exchange assumes that individual users and accounts are in a many-to-many relationship. Consider a household with three accounts:

- Checking 1 - held individually by one spouse
- Checking 2 - held jointly by both
- Checking 3 - held individually by the other spouse

Checking 2 should be available to either spouse, and the spouse holding Checking 1 should be able to see both Checking 1 and 2.

Open Financial Exchange expects FIs to give each user their own user ID and password. Each user will go through the enrollment step separately. A given account need only be activated once for a service; not once for each user. Clients will use the account information and activation messages to combine information about jointly-held accounts.

If an FI prefers to have a single user ID and password per household or per master account, they will have to make this clear to users through the enrollment process. It is up to the FI to assign a single user ID and password that can access all three of the checking accounts described above.

1. Enrollment and Password Acquisition <ENROLLRQ> <ENROLLRS>

The main purpose of the enrollment message is to communicate a user's intent to access the FI by way of Open Financial Exchange and to acquire a password for future use with Open Financial Exchange. Some FIs might return a user ID and an initial password in the enrollment response, while others will send them by way of regular mail.

NOTE: *Because the server does not know the user ID and password when the client sends the enrollment request, the <SONRQ> will not contain a valid user ID or password. The enrollment message accepts standard user identification information.*

Enrollment requests are not subject to synchronization. If the client does not receive a response, it will simply re-request the enrollment. If a user successfully enrolls from another client before the first client obtains a response, the server should respond to subsequent requests from the first client with status code:

```
13501 - user already enrolled.
```

1. User IDs

The Open Financial Exchange <SONRQ> requires a user ID to uniquely identify a user to an FI. Many FIs in the United States use social security numbers (SSNs) as the ID. Others create IDs that are unrelated to the users' SSNs. FIs can have an existing user IDs that they use for other online activities that they wish to use for Open Financial Exchange as well. They might also create new IDs specifically for Open Financial Exchange. Finally, some FIs might assign IDs while others might allow users to create them.

Because users do not usually know either their Open Financial Exchange sign-on user ID or their password at time of enrollment, the enrollment response is designed to return both. The enrollment request allows users to optionally provide a user ID, which an FI can interpret as their existing online ID or a suggestion for what their new user ID should be. It is recommended that the enrollment process explains ID syntax to users.

1. Enrollment Request

The enrollment request captures enough information to identify and authenticate a user as being legitimate and that it has a relationship with the FI.

FIs might require that an account number be entered as part of the identification process. However, this is discouraged since the account information request is designed to automatically obtain all account information, avoiding the effort and potential mistakes of a user-supplied account number.

It is **RECOMMENDED** that FIs provide detailed specifications for IDs and passwords along with information about the services available when a user is choosing an FI.

<ENROLLRQ>	Enrollment-request aggregate
<FIRSTNAME>	First name of user
<MIDDLENAME>	Middle name of user
<LASTNAME>	Last name of user
<ADDR1>	Address line 1
<ADDR2>	Address line 2
<ADDR3>	Address line 3
<CITY>	City
<STATE>	State or province
<POSTALCODE>	Postal code
<COUNTRY>	3-letter country code from ISO/DIS-3166
<DAYPHONE>	Daytime telephone number
<EVEPHONE>	Evening telephone number
<EMAIL>	Electronic e-mail address
<USERID>	Actual user ID if already known, or preferred user ID if user can pick
<TAXID>	ID used for tax purposes (such as SSN), may be same as user ID

<SECURITYNAME>	Mother's maiden name or equivalent
<DATEBIRTH>	Date of birth
<ACCTFROM>	An account description aggregate for one existing account at the FI, for identification purposes only. Can be <BANKACCTFROM>, <INVACCTFROM>, etc.
</ACCTFROM>	
</ENROLLRQ>	

This enrollment request is intended for use only by individuals. Business enrollment will be defined in a later release.

1. Enrollment Response

The main purpose of the enrollment response is to acknowledge the request. In those cases where FIs permit delivery of an ID and a temporary password, the response also provides for this. Otherwise the server will send the real response to the user by way of regular mail, electronic mail, or over the telephone. If enrollment is successful, but the server does not return the ID and password in the response, a server is REQUIRED to use status code 10 and provide some information to the user by means of the <MESSAGE> element in the <STATUS> aggregate about what to expect next.

<ENROLLRS>	Enrollment-response aggregate
<TEMPPASS>	Temporary password
<USERID>	User ID
<DTEXPIRE>	Time the temporary password expires (if <TEMPPASS> included)
</ENROLLRS>	

1. Enrollment Status Codes

0	Success (INFO)
2000	General error (ERROR)
13000	User ID & password will be sent out-of-band (INFO)
13500	Unable to enroll (ERROR)
13501	User already enrolled (ERROR)

1. Examples

An enrollment request:

```
<ENROLLTRNRQ>
        <TRNUID>12345
        <ENROLLRQ>
                <FIRSTNAME>Joe
                <MIDDLENAME>Lee
                <LASTNAME>Smith
                <ADDR1>21 Main St.
                <CITY>Anytown
                <STATE>TX
                <POSTALCODE>87321
                <COUNTRY>USA
                <DAYPHONE>123-456-7890
                <EVEPHONE>987-654-3210
                <EMAIL>jsmith@isp.com
                <USERID>jls
                <TAXID>123-456-1234
                <SECURITYNAME>jbmam
                <DATEBIRTH>19530202
        </ENROLLRQ>
</ENROLLTRNRQ>And the reply might be:
```

```
<ENROLLTRNRS>
    <TRNUID>12345
    <STATUS>
        <CODE>0
        <SEVERITY>INFO
    </STATUS>
    <ENROLLRS>
        <TEMPPASS>changeme
        <USERID>jls
        <DTEXPIRE>19970105
    </ENROLLRS>
</ENROLLTRNRS>
```

1. Account Information

Account information requests ask a server to identify and describe all of the accounts accessible by the signed-on user. The definition of *all* is up to the FI. At a minimum, it is **RECOMMENDED** that a server include information about all accounts that it can activate for one or more Open Financial Exchange services. To give the user a complete picture of his relationship with an FI, FIs can give information on other accounts, even if those accounts are available only for limited Open Financial Exchange services.

Some service providers will not have any prior knowledge of any user account information. The profile allows these servers to report this, and clients will then know to ask users for account information rather than reading it from the server.

Clients can perform several tasks for users with this account information. First, the information helps a client set up a user for online services by giving it a precise list of its account information and available services for each. Clients can set up their own internal state as well as prepare service activation requests with no further typing by users. This can eliminate data entry mistakes in account numbers, routing transit numbers, and so forth.

Second, FIs can provide limited information on accounts that would not ordinarily be suitable to Open Financial Exchange services. For example, a balance-only statement download would be useful for certificates of deposits even though a customer or an FI might not want or allow CDs to be used for full statement download.

For each account, there is one <ACCTINFO> aggregate returned. The aggregate includes one service-specific account information aggregate for each available service on that account. That, in turn, provides the service-specific account identification. Common to each service-specific account information aggregate is the <SVCSTATUS> tag, which indicates the status of this service on this account.

A server should return joint accounts (accounts for which more than one user ID can be used to access the account) for either user. Clients that wish to have a unified view will aggregate the results and remove duplicates before making specific requests involving joint accounts.

Requests and responses include a <DTACCTUP> element. Responses contain the last time a server updated the information. Clients can **OPTIONALLY** send this in a subsequent request, and servers are **REQUIRED** to compare this to the current modification time and only send information if it is more recent. The server sends the entire account information response if the client's time is older; there is no attempt to incrementally update specific account information.

1. Request <ACCTINFORQ>

<ACCTINFORQ>	Account-information-request aggregate
<DTACCTUP>	Last <DTACCTUP> received in a response
<INCIMAGES>	Y if server should include logo in response, N if client will separately fetch them based on given URL; *Boolean*
</ACCTINFORQ>	

1. Response <ACCTINFORS>

<ACCTINFORS>	Account-information-response aggregate
<DTACCTUP>	Date and time of last update to this information on the server
<ACCTINFO>	Zero or more account information aggregates
</ACCTINFO>	
</ACCTINFORS>	End of account information response

1. Account Information Aggregate <ACCTINFO>

<ACCTINFO>	Account-information-record aggregate
<DESC>	Description of the account, *A-80*
<PHONE>	Telephone number for the account, *A-20*
	URL to request the logo for the account

<LOGO>	(actual logos should be included via multi-part MIME in the response file if requested), *URL*
<XXXACCTINFO>	Service-specific account information, defined in each service chapter, one or more allowed
<XXXACCTFROM>	Service-specific account identification
</XXXACCTFROM>	
<SVCSTATUS>	AVAIL = Available, but not yet requested PEND = Requested, but not yet available ACTIVE = In use
</XXXACCTINFO>	
</ACCTINFO>	

NOTE: *A server uses the <DESC> field to convey the FI's preferred name for the account, such as "PowerChecking." It should not include the account number.*

1. Status Codes

0	Success (INFO)
2000	General error (ERROR)
13001	No change since supplied <DTACCTUP> (INFO)

1. ExamplesAn account information request:

```
<ACCTINFOTRNRQ>
        <TRNUID>12345
        <ACCTINFORQ>
                <DTACCTUP>19960101
                <INCIMAGES>N
        </ACCTINFORQ>
</ACCTINFOTRNRQ>And a response for a
user with access to one account, supporting banking:
<ACCTINFOTRNRS>
        <TRNUID>12345
        <STATUS>
                <CODE>0
                <SEVERITY>INFO
        </STATUS>
        <ACCTINFORS>
                <DTACCTUP>19960102
                <ACCTINFO>
                <DESC>Power Checking
                <PHONE>8002223333
                <LOGO>https://www.fi.com/ofx/logos/powercheck.jpg
                        <BANKACCTINFO>
                                <BANKACCTFROM>
```

```
                                        <BANKID>1234567789
                                        <ACCTID>12345
                                        <ACCTTYPE>CHECKING
                                </BANKACCTFROM>
                        <SUPTXDL>Y
                        <XFERSRC>Y
                        <XFERDEST>Y
                        <SVCSTATUS>ACTIVE
                        </BANKACCTINFO>
                </ACCTINFO>
        </ACCTINFORS>
</ACCTINFOTRNRS>
```

1. Service Activation

Clients inform FIs that they wish to start, modify, or terminate a service for an account by sending service activation requests. These are subject to data synchronization, and servers should send responses to inform clients of any changes, even if the changes originated on the server.

Clients use these records during the initial user sign-up process. Once a client learns about the available accounts and services (by using the account information request above, or by having a user directly enter the required information), it sends a series of service ADD requests.

If a user changes any of the identifying information about an account, the client sends a service activation request containing both the old and the new account information. Servers should interpret this as a change in the account, not a request to transfer the service between two existing accounts, and all account-based information such as synchronization tokens should continue. If a user or FI is reporting that service should be moved between two existing accounts, service must be terminated for the old account and started for the new account. The new account will have reset token histories, as with any new service.

Each service to be added, changed, or removed is contained in its own request because the same real-world account might require different <ACCTFROM> aggregates depending on the type of service.

1. Activation Request and Response

2. Request <ACCTRQ>

<ACCTRQ>	Account-service-request aggregate
<ACTION>	Action aggregate, either <SVCADD>, <SVCCHG>, or <SVCDEL>
</ACTION>	
<SVC>	Service to be added/changed/deleted
</ACCTRQ>	

1. Response <ACCTRS>

<ACCTRS>	Account-service-response aggregate
<ACTION>	Action aggregate, either <SVCADD>, <SVCCHG>, or <SVCDEL>
</ACTION>	

<SVC>	Service to be added/changed:
	
	BANKSVC = Banking service
	BPSVC = Payments service
	INVSVC = Investments

</ACCTRS>	

1. Service Add Aggregate <SVCADD>

<SVCADD>	Service-add aggregate
<ACCTTO>	Service-specific-account-identification aggregate (see <BANKACCTTO>, <INVACCTTO>)
</ACCTTO>	
</SVCADD>	

1. Service Change Aggregate <SVCCHG>

<SVCCHG>	Service-add aggregate
<ACCTFROM>	Service-specific-account-identification aggregate (see <BANKACCTFROM>, <INVACCTFROM>)
</ACCTFROM>	
<ACCTTO>	Service-specific-account-identification aggregate (see <BANKACCTTO>, <INVACCTTO>)
</ACCTTO>	
</SVCCHG>	

1. Service Delete Aggregate <SVCDEL>

<SVCDEL>	Service-deletion aggregate
<ACCTFROM>	Service-specific-account-identification aggregate (see <BANKACCTFROM>, <INVACCTFROM>)
</ACCTFROM>	

</SVCDEL>	

1. Status Codes

0	Success (INFO)
2000	General error (ERROR)
2002	Other account error (ERROR)
2006	Source (from) account not found (ERROR)
2007	Source (from) account closed (ERROR)
2008	Source (from) account not authorized (ERROR)
2009	Destination (to) account not found (ERROR)
2010	Destination (to) account closed (ERROR)
2011	Destination (to) account not authorized (ERROR)
13502	Invalid service (ERROR)

1. Service Activation Synchronization

Service activation requests are subject to the standard data synchronization protocol. The scope of these requests and the <TOKEN> is the user-ID. The request and response tags are <ACCTSYNCRQ> and <ACCTSYNCRS>.

1. Examples

Activating a payment:

```
<ACCTTRNRQ>
        <TRNUID>12345
        <ACCTRQ>
                <SVCADD>
                        <BANKACCTTO>
                                <BANKID>1234567789
                                <ACCTID>12345
                                <ACCTTYPE>CHECKING
                        </BANKACCTTO>
                </SVCADD>
                <SVC>BPSVC
        </ACCTRQ>
</ACCTTRNRQ>
```

A response:

```
<ACCTTRNRS>
    <TRNUID>12345
        <STATUS>
        <CODE>0
        <SEVERITY>INFO
    </STATUS>
    <ACCTRS>
        <SVCADD>
            <BANKACCTTO>
                <BANKID>1234567789
                <ACCTID>12345
                <ACCTTYPE>CHECKING
```

```
                    </BANKACCTTO>
              </SVCADD>
              <SVC>BPSVC
        </ACCTRS>
</ACCTTRNRS>
```

1. Name and Address Changes <CHGUSERINFORQ> <CHGUSERINFORS>

Users may request that an FI update the official name, address, phone, and e-mail information using the <CHGUSERINFORQ>. Only the fields that should be changed are sent. The response reports all of the current values. For security reasons, some of the fields in the <ENROLLRQ> cannot be changed online, such as tax ID.

The transaction tag is <CHGUSERINFOTRNRQ> and <CHGUSERINFOTRNRSRQ>. These methods are subject to synchronization, <CHGUSERINFOSYNCRQ> and <CHGUSERINFOSYNCRS>.

1. <CHGUSERINFORQ>

<CHGUSERINFORQ>	Change-user-information-request aggregate
<FIRSTNAME>	First name of user
<MIDDLENAME>	Middle name of user
<LASTNAME>	Last name of user
<ADDR1>	Address line 1
<ADDR2>	Address line 2
<ADDR3>	Address line 3
<CITY>	City
<STATE>	State or province
<POSTALCODE>	Postal code
<COUNTRY>	3-letter country code from ISO/DIS-3166
<DAYPHONE>	Daytime telephone number
<EVEPHONE>	Evening telephone number
<EMAIL>	Electronic e-mail address
</CHGUSERINFORQ>	

1. <CHGUSERINFORS>

<CHGUSERINFORS>	Change-user-information-request aggregate
<FIRSTNAME>	First name of user

<MIDDLENAME>	Middle name of user
<LASTNAME>	Last name of user
<ADDR1>	Address line 1
<ADDR2>	Address line 2
<CITY>	City
<STATE>	State or province
<POSTALCODE>	Postal code
<COUNTRY>	3-letter country code from ISO/DIS-3166
<DAYPHONE>	Daytime telephone number
<EVEPHONE>	Evening telephone number
<EMAIL>	Electronic e-mail address
<DTINFOCHG>	Date and time of update *datetime*
</CHGUSERINFORS>	

1. Status Codes

0	Success (INFO)
2000	General error (ERROR)
13503	Cannot change user information (ERROR)

1. Signup Message Set Profile Information

A server must include the following aggregates as part of the profile <MSGSETLIST> response, since every server must support at least the account information and service activation messages. In the <ENROLLPROF> aggregate, servers indicate how enrollment should proceed: via the client, a given web page, or a text message directing users to some other method (such as a phone call)..

<SIGNUPMSGSET>	Signup-message-set-profile-information aggregate
<SIGNUPMSGSETV1>	Opening tag for V1 of the message set profile information
<MSGSETCORE>	Common message set information, defined in the profile chapter
</MSGSETCORE>	
	Enrollment options - only one of <CLIENTENROLL>, <WEBENROLL>, or <OTHERENROLL> is allowed

<CLIENTENROLL>	Client-based enrollment supported
<ACCTREQUIRED>	Y if account number is required as part of enrollment *Boolean*
</CLIENTENROLL>	
<WEBENROLL>	Web-based enrollment supported
<URL>	URL to start enrollment process
</WEBENROLL>	
<OTHERENROLL>	Some other enrollment process
<MESSAGE>	Message to give to consumer about what to do next (e.g. a phone number) *A-80*
</OTHERENROLL>	
<CHGUSERINFO>	Y if server supports client-based user information changes
<AVAILACCTS>	Y if server can provide information on accounts with SVCSTATUS available, N means client should expect to ask user for specific account information *Boolean*
</SIGNUPMSGSETV1>	
</SIGNUPMSGSET>	

1. Customer to FI Communication

2. The E-Mail Message Set

The e-mail message set includes two messages: generic e-mail and generic MIME requests by way of URLs. In Open Financial Exchange files, the message set name is EMAILMSGSV1.

1. E-Mail Messages

Open Financial Exchange allows consumers and FIs to exchange messages. The message body is in HTML so that FIs can provide some graphic structure to the message. Keep in mind that, as with regular World Wide Web browsing, an Open Financial Exchange client might not support some or all of the HTML formatting, so the text of the message must be clear on its own. Clients can request that graphics (the images referenced in an <IMG> tag) be sent as part of the response file, or clients can separately request those elements. If a server sends images, it should use the standard procedure for incorporating external data as described in Chapter 2. Servers are not required to support HTML or to send images, even if the client asks.

A user or an FI can originate a message. E-mail messages are subject to data synchronization so that a server can send a response again if it is lost or if it is used by multiple clients.

Because e-mail messages cannot be replied to immediately, the response should just echo back the original message (so that data synchronization will get this original e-mail message to other clients). When the FI is ready to reply, it should generate an unsolicited response (<TRNUID>0) and the client will pick this up during synchronization.

Client Sends	*Server Responds*
Account information	
From, To	
Subject	
Message	
	Account information
	From, To
	Subject
	Message
	Type

1. Regular vs. Specialized E-Mail

Several services with Open Financial Exchange define e-mail requests and responses that contain additional information specific to that service. To simplify implementation for both clients and servers, this section defines a <MAIL> aggregate that Open Financial Exchange uses in all e-mail requests and responses. For regular e-mail, the only additional information is an account from aggregate and whether to include images in the e-mail response or not.

1. Basic <MAIL> Aggregate

<MAIL>	Core e-mail aggregate
<USERID>	User ID such as SSN
<DTCREATED>	When message was created *datetime*
<FROM>	Customer's input for whom message is from, *A-32*
<TO>	Who e-mail should be delivered to, *A-32*
<SUBJECT>	Subject of message (plain text, not HTML), *A-60*
<MSGBODY>	Body of message, HTML-encoded or plain text depending on <USEHTML>, *A-10000*
</MSGBODY>	End of message

<INCIMAGES>	Include images in response, *Boolean*
<USEHTML>	Y if client wants an HTML response, N if client wants plain text, *Boolean*
</MAIL>	

If using HTML for the message body, clients and servers are **REQUIRED** to wrap the desired HTML in an SGML marked section to protect the HTML markup: <![CDATA [... html ...]]>. See the example.

1. E-Mail <MAILRQ> <MAILRS>

E-mail is subject to synchronization. The transaction tag is <MAILTRNRQ> / <MAILTRNRS> and the synchronization tag is <MAILSYNCRQ> / <MAILSYNCRS>.

<MAILRQ>	E-mail-message-request aggregate
<MAIL>	Core e-mail aggregate
</MAIL>	
</MAILRQ>	

In a response, the <TRNUID> is zero if this is an unsolicited message. Otherwise, it should contain the <TRNUID> of the user's original message. It is RECOMMENDED that servers include the <MESSAGE> of the user's message as part of the reply <MESSAGE>. The <MESSAGE> contents can include carriage returns to identify desired line breaks.

<MAILRS>	E-mail-message-response aggregate
<MAIL>	Core e-mail aggregate
</MAIL>	
</MAILRS>	

1. Status Codes

0	Success (INFO)
2000	General error (ERROR)
16500	HTML not allowed (ERROR)
16501	Unknown mail To: (ERROR)

1. E-Mail Synchronization <MAILSYNCRQ> <MAILSYNCRS>

Open Financial Exchange uses data synchronization to collect responses that could have been lost due to communication problems, or that the servers previously sent to a different client or data file. All messages sent to the signed-on user ID are covered by a single <TOKEN>. Note that this synchronization action expects only the basic <MAILRS> responses. Specialized e-mail is received by means of their own synchronization requests.

<MAILSYNCRQ>	E-mail-synchronization-request aggregate
<TOKEN>	Client history marker
<INCIMAGES>	Include images in response, *Boolean*
<USEHTML>	Y if client wants an HTML response, N if client wants plain text, *Boolean*
</MAILSYNCRQ>	

<MAILSYNCRS>	E-mail-synchronization-response. aggregate
<TOKEN>	Server history marker
<MAILTRNRS>	Missing e-mail response transactions (0 or more)
</MAILSYNCRS>	

1. Example

In this example, a consumer requests information from customer service about the checking statement just downloaded. This example omits the <OFX> top level and the signon <SONRQ>. This example uses HTML for the message body, and so it must protect the HTML content in an SGML CDATA marked section.The request:

```
<MAILTRNRQ>
        <TRNUID>54321
        <MAILRQ>
                <MAIL>
                        <USERID>123456789
                        <FROM>James Hackleman
                        <TO>Noelani Federal Savings
                        <SUBJECT>What do I need to earn interest?
                        <DTCREATED>19960305
                           <MSGBODY><![ CDATA [<HTML><BODY>I didn't earn any interest this month. Can yo
]]></MSGBODY>
                           <INCIMAGES>N
                           <USEHTML>Y
                  </MAIL>
         </MAILRQ>
</MAILTRNRQ>The response from the FI:
<MAILTRNRS>
     <TRNUID>54321
     <STATUS>
          <CODE>0
          <SEVERITY>INFO
          </STATUS>
     <MAILRS>
          <MAIL>
               <USERID>123456789
               <DTCREATED>19960307
               <FROM>Noelani Federal Savings
               <TO>James Hackleman
               <SUBJECT>Re: What do I need to earn interest?
                         <MSGBODY>><![ CDATA [<HTML><BODY>You need to maintain $1000 in this account t
Sincerely,
Customer Service Department

Original message:
I didn't earn any interest this month. Can you please tell me what I need to do to earn interest on t
]]></MSGBODY>
                         <INCIMAGES>N
                         <USEHTML>Y
                 </MAIL>
        </MAILRS>
</MAILTRNRS>
```

1. Example of Synchronization Involving E-Mail

In the following example the client did not receive the reply to the message sent in the previous example, so its <TOKEN> is one less than the server's. The server replies by giving the current <TOKEN> and the missed response.

```
<MAILSYNCRQ>
        <TOKEN>101
</MAILSYNCRQ>

<MAILSYNCRS>
        <TOKEN>102
        <MAILTRNRS>
                <TRNUID>54321
                <STATUS>
                        <CODE>0
                        <SEVERITY>INFO
                </STATUS>
                <MAILRS>
                        ... contents of e-mail message response as shown in previous example
                </MAILRS>
        </MAILTRNRS>
</MAILSYNCRS>
```

1. Get HTML Page

Some responses contain values that are URLs, intended to be separately fetched by clients if desired. Clients can use their own HTTP libraries to perform this fetch outside of the Open Financial Exchange specification. However, to insulate clients against changes in transport technology, and to allow for fetches that require the protection of an authenticated signon by a specific user, Open Financial Exchange defines a transaction roughly equivalent to an HTTP Get. Any MIME type can be retrieved, including images as well as HTML pages.

1. MIME Get Request and Response <GETMIMERQ> <GETMIMERS>

The following table lists the components of a request:

<GETMIMERQ>	Get-MIME-request aggregate
<URL>	URL, *URL*
</GETMIMERQ>	

The response simply echoes back the URL. The actual response, whether HTML, an image, or some other type, is always sent as a separate part of the file using multi-part MIME.

<GETMIMERS>	Get-MIME-response aggregate
<URL>	URL, *URL*
</GETMIMERS>	

1. Status Codes

0	Success (INFO)
2000	General error (ERROR)
2019	Duplicate transaction (ERROR)
16502	Invalid URL (ERROR)
16503	Unable to get URL (ERROR)

1. Example A request:

```
<GETMIMETRNRQ>
        <TRNUID>54321
        <GETMIMERQ>
                <URL>http://www.fi.com/apage.html
        </GETMIMERQ>
</GETMIMETRNRQ>A response - the full
file is shown here to illustrate the use of multi-part MIME:
HTTP 1.0 200 OK
Content-Type: multipart/x-mixed-replace; boundary =--boundary-

--boundary--
Content-Type: application/x-ofx
Content-Length: 8732

OFXHEADER:100
DATA:OFXSGML
VERSION:100
ENCRYPTION:1
ENCODING:USASCII

<OFX>
                  <!-- signon not shown
                  message set wrappers not shown -->
<GETMIMETRNRS>
```

```
        <TRNUID>54321
        <STATUS>
                <CODE>0
                <SEVERITY>INFO
        </STATUS>
        <GETMIMERS>
                <URL>http://www.fi.com/apage.html
        </GETMIMERS>
</GETMIMETRNRS>
</OFX>

--boundary--
Content-Type: text/html
<HTML>
        <!-- standard HTML page -->
</HTML>

--boundary--
```

1. E-Mail Message Set Profile Information

If either or both of the messages in the e-mail message set are supported, the following aggregate must be included in the profile <MSGSETLIST> response.

<EMAILMSGSET>	E-mail-message-set-profile-information aggregate
<EMAILMSGSETV1>	Opening tag for V1 of the message set profile information
<MSGSETCORE>	Common message set information, defined in the profile chapter
</MSGSETCORE>	
<EMAIL>	Y if server supports generic e-mail message
<GETMIME>	Y if server supports get MIME message
</EMAILMSGSETV1>	
</EMAILMSGSET>	

1. Recurring Transactions

Open Financial Exchange enables users to automate transactions that occur on a regular basis. Recurring transactions are useful when a customer has payments or transfers, for example, that repeat at regular intervals. The customer can create a "model" at the server for automatic generation of these instructions. The model in turn creates payments or transfers until it is canceled or expires. After the user creates a recurring model at the server, the server can relieve the user from the burden of creating these transactions; it generates the transactions on its own, based on the operating parameters of the model.

1. Creating a Recurring Model

The client must provide the following information to create a model:

- Type of transaction generated by the model (payment or transfer)
- Frequency of recurring transaction
- Total number of recurring transactions to generate
- Service-specific information, such as transfer date, payment amount, payee address

The model creates each transaction some time before its due date, usually thirty days. This allows the user to retrieve the transactions in advance of posting. This also gives the user the opportunity to modify or cancel individual transactions without changing the recurring model itself.

When a model is created, it can generate several transactions immediately. The model does not automatically return responses for the newly created transactions. It only returns a response to the request that was made to create the model. For this reason, clients should send a synchronization request along with the request to create a model. This allows the server to return the newly created transaction responses, as well as the response to the request to set up a new model.

1. Recurring Instructions <RECURRINST>

The Recurring Instructions aggregate is used to specify the schedule for a repeating instruction. It is passed to the server when a recurring transfer or payment model is first created.

<RECURRINST>	Recurring-Instructions aggregate
<FREQ>	Frequency, see section 10.2.1
<NINSTS>	Number of instructions If this tag is absent, the schedule is open-ended, *N-3*
</RECURRINST>	

1. Values for <FREQ>

WEEKLY	Weekly
BIWEEKLY	Biweekly

TWICEMONTHLY	Twice a month
MONTHLY	Monthly
FOURWEEKS	Every four weeks
BIMONTHLY	Bimonthly
QUARTERLY	Quarterly
SEMIANNUALLY	Semiannually
TRIANNUALLY	Triannually
ANNUALLY	Annually

Rules for calculating recurring dates of WEEKLY, BIWEEKLY, and TWICEMONTHLY are as follows:

- WEEKLY = starting date for first transaction, starting date + 7 days for the second
- TWICEMONTHLY = starting date for first, starting date + 15 days for the second
- BIWEEKLY = starting date for first, starting date + 14 days for the second

Examples:

Start date of May 2: next transaction date for WEEKLY is May 9; TWICEMONTHLY is May 17; next transfer date for BIWEEKLY is May 16.

Start date of May 20: next date for WEEKLY is May 27; TWICEMONTHLY is June 4; next date for BIWEEKLY is June 3.

TWICEMONTHLY recurring transactions will occur each month on those days adjusting for weekends and holidays. BIWEEKLY will occur every 14 days.

1. Examples

The following example illustrates the creation of a repeating payment. The payment repeats on a monthly basis for 12 months. All payments are for $395.The request:

```
.
.
.
<RECPMTRQ>
        <RECURRINST>
                <FREQ>MONTHLY
                <NINSTS>12
        </RECURRINST>
        <PMTINFO>
                <BANKACCTFROM>
```

```
                        <BANKID>555432180
                        <ACCTID>763984
                        <ACCTTYPE>CHECKING
                </BANKACCTFROM>
                <TRNAMT>395.00
                <PAYEEID>77810
                <PAYACCT>444-78-97572
                <DTDUE>19971115
                <MEMO>Auto loan payment
        </PMTINFO>
</RECPMTRQ>
.
.
.

The response includes the <RECSRVRTID>
that the client can use

to cancel or modify the model:
.
.
.
<RECPMTRS>
        <RECSRVRTID>387687138
        <RECURRINST>
                <FREQ>MONTHLY
                <NINSTS>12
        </RECURRINST>
        <PMTINFO>
                <BANKACCTFROM>
                        <BANKID>555432180
                        <ACCTID>763984
                        <ACCTTYPE>CHECKING
                </BANKACCTFROM>
                <TRNAMT>395.00
                <PAYEEID>77810
                <PAYACCT>444-78-97572
                <DTDUE>19971115
                <MEMO>Auto loan payment
        </PMTINFO>
</RECPMTRS>
.
.
.
```

1. Retrieving Transactions Generated by a Recurring Model

Once created, a recurring model independently generates instructions. Since the client has not directly generated these transactions, the client has no record of their creation. To enable users to modify and/or cancel pending instructions, the client must use data synchronization in order to retrieve these transactions.

The client has two purposes for synchronizing state with the server with respect to recurring models:

- Retrieve any added, modified, or canceled recurring models
- Retrieve any added, modified, or canceled transactions generated by any models

The client must be able to synchronize with the state of any models at the server, as well as the state of any transactions generated by the server.

1. Modifying and Canceling Individual Transactions

Once created and retrieved by the customer, recurring payments and transfers are almost identical to customer-created payments or transfers. As with ordinary payments or transfers, you can cancel or modify transactions individually. However, because servers generate these transfers, they are different in the following respects:

- Recurring transactions must be retrieved as part of a synchronization request.
- Recurring transactions are related to a model. A server can modify or cancel transactions if the model is modified or canceled.

1. Modifying and Canceling Recurring Models

A recurring model can be modified or canceled. When a model is modified, all transactions that it generates in the future will change as well. The client can indicate whether transactions that have been generated, but have not been sent, should be modified as well. The actual elements within a transaction that can be modified differ by service. See the recurring sections within the Banking and Payments chapters for details.

A user can cancel a model immediately or at a future date. If a user cancels the model immediately, the client cancels any transactions that it has not yet sent. If the client schedules the cancel for a future date, the client will not cancel pending transactions.

1. Examples

Canceling a recurring payment model requires the client to pass the <RECSRVRTID> of the model. The client requests that pending payments also be canceled. The server cancels the model immediately and notifies the client that both the model and any scheduled payments were canceled.The request:

```
.
.
.
        <RECPMTCANCRQ>
                <RECSRVRTID>387687138
                <CANPENDING>Y
        </RECPMTCANCRQ>
.
.
.
The response:
.
.
.
    <RECPMTCANCRS>
        <RECSRVRTID>387687138
        <CANPENDING>Y
    </RECPMTCANCRS>
.
.
```

.

stadyn_image9.gif (7739 bytes)

stadyn_image9

The server also cancels any payments that have been generated but not executed. In the example shown above, the client would not learn of this immediately. To receive notification that the model and all generated payments were canceled, the client would need to include a synchronization request in the file. The following example illustrates this alternate approach.The request file now includes a synchronization request:

```
.
.
.
        <RECPMTCANCRQ>
                <RECSRVRTID>387687138
                <CANPENDING>Y
        </RECPMTCANCRQ>
        <PMTSYNCRQ>
                <TOKEN>12345
                <BANKACCTFROM>
                        <BANKID>123432123
                        <ACCTID>516273
                        <ACCTTYPE>CHECKING
                </BANKACCTFROM>
        </PMTSYNCRQ>
.
.
.
The response file now contains two responses
(assuming one payment was pending),

one for the canceled model and one for the canceled payment.
.
.
.
```

```
    <RECPMTCANCRS>
        <RECSRVRTID>387687138
        <CANPENDING>Y
    </RECPMTCANCRS>
    <PMTSYNCRS>
        <TOKEN>3247989384
        <BANKACCTFROM>
            <BANKID>123432123
            <ACCTID>516273
            <ACCTTYPE>CHECKING
        </BANKACCTFROM>
        <PMTTRNRS>
```

```
            <TRNUID>10103
            <STATUS>
                <CODE>0
                <SEVERITY>INFO
            </STATUS>
            <PMTCANCRS>
                <SRVRTID>1030155
            </PMTCANCRS>
        </PMTTRNRS>
    </PMTSYNCRS>
.
.
.
```

stadyn_image10