FXintegrate Real-Time Programming Guide

Version 1.1 July 18, 2005

Currenex, Incorporated
3565 Haven Avenue
Menlo Park, California 94025

FXINTEGRATE REAL-TIME PROGRAMMING GUIDE 1

1. Introduction

FXintegrate Real-Time is a seamless post-trade integration service for foreign exchange (FX) straight through processing (STP). As an automated, real-time trade capturing service, Real-Time significantly increasing FX trading efficiency eliminating the delays and errors associated with manually transferring completed trade details from one system to another.

The FXintegrate Real-Time service enables Currenex customers to receive trade details in real-time as the trades are executed on the Currenex FXtrades service. A flexible and secure interface, FXintegrate Real-Time uses industry standard XML file formatting and web-based services to provide timely and accurate information to both buy and sell side customers.

1.1 Document Scope

This programming guide is intended for customers implementing HTTPS software to receive completed trade and settlement information postings from the Currenex FXintegrate Real-Time service. It is an in depth manual that explains all the functions and components of the FXintegrate Real-Time service and presents examples.

Familiarity with the concept of application programming, data communication, data security, digital certificates, XML and the HTTPS protocol is needed.

2. System Architecture

2.1 Overview

FXintegrate Real-Time (FXRT) is a web application that sends XML based post-trade information to customers over the public Internet or private circuits.

FXRT writes or "pushes" completed trade details to a customer specified web address as soon as a trade is booked on FXtrades. Any web technology that conforms to Currenex requirements can be used to receive the pushed trade.

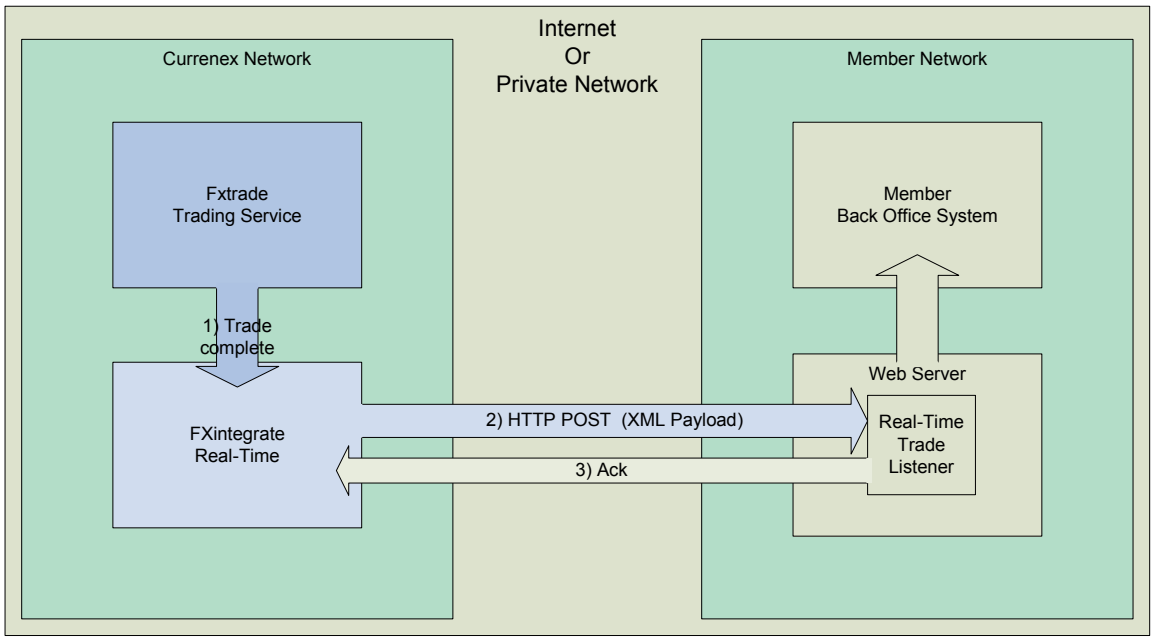

Figure 1: FXintegrate Real-Time Architecture.

As illustrated in Figure 1 above, Currenex posts completed trade details to a Real-Time Trade Listener (RTTL) residing on a customer side web server. The RTTL can be developed using a Java servlet, a Common Gateway Interface (CGI), or a PHP application

Customers not requiring post trade information in real-time can use the Currenex FXintegrate On-Demand product, which uses a polling model to deliver completed trade details to customers. On-Demand does not require a customer side service to receive the data. Refer to the FXintegrate On-Demand Developer's Guide for more details.

3. Trade Delivery

3.1 Delivery Mechanism

FXRT delivers trade information over the network using standard HTTPS POST. The customer must have a service listening on a specific address and port in order to receive this information. Currenex does not impose any technology standard on the customer. As long as the service can properly accept and process an incoming HTTPS POST, a customer can use any technology that fits its infrastructure.

The HTTPS POST payload is in XML format. Currenex has developed a DTD/Schema for its trade-capture service that defines all the field syntax within the XML data. The DTD/Schema is provided under separate cover.

Currenex requires the following for the https server setup:

- An https server with mutual SSL connection configured.
- The https server must be signed by a commercial trust provider, e.g., Verisign, Equifax, etc.
- The Currenex Real-Time certificate installed as a trusted certificate authority into the client's web server.

3.2 Payload

The payload uses standard HTTP. It can be either the trade request message or the reply message in XML. The customer side software determines how the push information is extracted and how the reply is sent. If using a servlet, the standard servlet interface contains the appropriate interfaces.

3.3 Post Payload

3.3.1 Post

Each POST to the user specified URL has a content type of "text/XML." The content syntax is defined by the CurrenexDownload.dtd or xsd, which is maintained undr separate cover.

The following post is intended as an example. The user's actual post will differ.

```
POST /trades/tradehandler HTTP/1.1
Host: fxtrades.currenex.net
User-Agent: Apache
Content-Length: 2345
Content-Type: text/XML

<trade>
      <tradeHeader>
           <partyTradeIdentifier>
                <partyReference href="REXXXXX_C"/>
                <tradeId>78369-16881</tradeId> ClOrdId
           </partyTradeIdentifier>
           <partyTradeIdentifier>
                <partyReference href="Currenex_X"/>
                <tradeId>A200515406XB000</tradeId> ExecId
                <linkId>
                     <tradeId>A200515406XC000</tradeId>
                     <linkIdType>OrderToOrder</linkIdType>
                </linkId>
.
.
.
```

The entire HTTP post consists of the sent trade data in XML.

The below java source file shows how a client can retrieve the entire post and construct a proper reply.

```
package currenex.server.stp;
```

```
import java.io.IOException;
import java.util.logging.Logger;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletResponse;

/**
 * For testing RTTC.
 *
 * @author XXXX
 * @version $Revision: #2 $, $DateTime: 2005/03/17 16:32:54 $
 */
public final class T_ClientServlet extends
javax.servlet.http.HttpServlet
{
    private static final Logger s_log =
Logger.getLogger(T_ClientServlet.class.getName());
    private static final java.util.Random s_random = new
java.util.Random();

    public void doPost(HttpServletRequest request,
HttpServletResponse response)
        throws ServletException, IOException
    {
        String contentType = request.getContentType();
        int contentLength = request.getContentLength();
        System.out.println("doPost - content="+contentType+"
length="+contentLength
                +" remoteAddr="+request.getRemoteAddr());
        byte[] content = new byte[contentLength];
        java.io.InputStream in = request.getInputStream();
        int bytesRead = 0;
        while (bytesRead < contentLength)
        {
            int ret = in.read(content, bytesRead, content.length
- bytesRead);
            if (ret < 0) break;
            else bytesRead += ret;
        }
        String body = new String(content);
        s_log.fine("body: "+body);

        java.io.PrintWriter out = response.getWriter();
        if ("text/xml".equals(contentType))
        { // RTTC
            System.out.println("cx xml: "+body);
            int beginIdx = body.indexOf("<tradeId>");
            if (beginIdx < 0) throw new ServletException("tradeId
not found");
            beginIdx += "<tradeId>".length();
            int endIdx = body.indexOf("</tradeId>", beginIdx);
            String tradeId = body.substring(beginIdx, endIdx);
            String status = (s_random.nextBoolean() ? "received"
: "error");
```

```
            System.out.println("doPost - responding to RTTC
"+tradeId+" with "+status);
            out.println("<?xml version=\"1.0\"?>");

out.println("<RealTimeReply><trade><id>"+tradeId+"</id><status>"+
status+"</status></trade></RealTimeReply>");
        }
        else throw new ServletException("unrecognized
contentType: "+contentType);
        out.flush();
    }
}
```

3.3.2 Reply

The reply message should be structured as follows. The actual trade ids will vary.

```
<?xml version="1.0" encoding="UTF-8"?>
<RealTimeReply>
<trade>
   <id>A2004304005X000</id>
   <status>received</status>
</trade>
<trade>
   <id>A2004304006X000</id>
   <status>received</status>
</trade>
<trade>
   <id>A2004304007X000</id>
   <status>received</status>
</trade>

</RealTimeReply>
```

The actual trade id being replied to appears between the <id> and </id> tags. For each trade message Currenex pushes, a client should reply with the above message. Upon receipt of the reply, Currenex will mark the trade as received. Otherwise, the trade will be left as active if any other status or if no reply is received. Currenex continues to push trades even during its nightly recovery interval.

Generally, a post will contain only one trade. However, if there is a disconnect, posts will queued until the connection is re-established at which point all queued trades will be pushed in a single post.

If an XML post is received containing more than one trade, a reply must be sent for each trade processed. For example, if a post contains 10 trades and all 10 trades are processed, the reply would need to contain 10 separate <trade> tags, one for each trade.

3.3.3 Sample Post XML data

A sample Post XML data section appears below. The <tradeId> and </tradeId> tags within the <linked> </linked> tag is used by Prime Brokers. All other customers should use the <tradeId> under <partyTradeIdentifier> as the final executed trade id.

```
<trade>
<tradeHeader>
   <partyTradeIdentifier>
      <partyReference href="REXXXXX_C"/>
      <tradeId>78369-16881</tradeId> ClOrdId
   </partyTradeIdentifier>
   <partyTradeIdentifier>
      <partyReference href="Currenex_X"/>
      <tradeId>A200515406XB000</tradeId> ExecId
      <linkId>
         <tradeId>A200515406XC000</tradeId>
         <linkIdType>OrderToOrder</linkIdType>
      </linkId>
   </partyTradeIdentifier>
   <tradeDateTime>2005-05-24T13:44:26Z</tradeDateTime>
   <tradeStatus>DONE</tradeStatus>
   <trader>
      <partyReference href=" REXXXXX_C"/>
      <userId>FX3663</userId>
   </trader>
   <trader>
      <partyReference href=" REXXXXPB_B"/>
      <userId>admin1</userId>
   </trader>
   <subFund>REFX3660</subFund>
   <event>
      <eventType>NEWT</eventType>
      <eventDateTime>2005-05-24T13:44:26Z</eventDateTime>
   </event>
</tradeHeader>
<tradeRequest>
   <requesterPartyReference href=" REXXXXX_C"/>
   <buySell>SELL</buySell>
   <specifiedMoney>
      <currency>EUR</currency>
      <amount>100000.00</amount>
   </specifiedMoney>
   <againstCurrency>USD</againstCurrency>
   <tenor>
      <period>SP</period>
   </tenor>
   <deliveryType>DEL</deliveryType>
</tradeRequest>
<product>
   <productType>SP</productType>
   <fxLeg>
      <cashFlow1>
```

```
        <currency>USD</currency>
        <amount>126020.00</amount>
        <buyerPartyReference href=" REXXXXX_C"/>
      </cashFlow1>
      <cashFlow2>
        <currency>EUR</currency>
        <amount>100000.00</amount>
        <buyerPartyReference href=" REXXXXXPB_B"/>
      </cashFlow2>
      <valueDate>2005-05-26</valueDate>
      <exchangeRate>
        <currency1>USD</currency1>
        <currency2>EUR</currency2>
        <quoteBasis>currency1percurrency2</quoteBasis>
        <rate>1.26020000</rate>
        <spotRate>1.26020000</spotRate>
        <points>0.00000000</points>
      </exchangeRate>
    </fxLeg>
</product>
```

3.4 Security

For secure communication and authentication, FXintegrate Real-Time relies on the secured hypertext transfer protocol. FXintegrate Real-Time establishes its connection via HTTP over SSL on the standard default port of 443. The Currenex server is configured for mutual authentication via digital certificates.

In order to connect to the FXintegrate Real-Time service, the connecting member must present a valid server certificate that is issued by the Currenex Certification Authority (CA).

4. Business Process Architecture

4.1 Real-Time Trade Events

Real-Time trade events are created by transitions in the trade lifecycle. These events are used by the Real-Time service to communicate trade changes to a receiving customer. Each trade has two counterparties and either can precipitate trade events. The actions of one party do not affect the other.

Each event has a type that reflects the nature of the transition a trade has experienced.

STP Events	
Event Code	**Description**
NEWT	This event is created when a trade is executed as the result of a price discovery process. The event can occur at most once in the history of a trade and may never occur (see the following). This event is created when a trade is executed or when an order is filled.
AMND	This event occurs when an existing trade is amended. It will have a new trade ID and will contain a reference back to the original trade.
CANC	This event is created when an existing trade is cancelled outright. The trade ID in this event will refer to the trade being cancelled. Once a trade has been cancelled no other events can occur on the trade.
ROLL	This event occurs when a trade is rolled forward. Each ROLL contains a new, unique trade ID and a reference back to the original trade. A trade may be rolled forward into more than one new trade. For each of these new trades a ROLL event will be generated that refers back to the original trade. It is possible for rolled trades to in turn be rolled.
ALOC	This event is similar to the ROLL in that it is one of many children of a parent trade. ALOC events occur when a trade is allocated. As with ROLL events each one will contain a unique trade ID and a reference to the originating trade.
UDFA	Defined fields can be altered anytime during the history of a trade. When they are altered a UDFA event records the fact that the change occurred. Many UDFA events can occur on a trade. Each UDFA event will contain the trade ID of the trade being modified. New trade ID's are not assigned for a defined field change.
CNDF	This is a special event necessitated by the behavior of defined fields. The letters stand for *cancelled with incomplete defined fields*. See the following discussion for further explanation.
SETL	This event occurs when the settlement instructions have been finalized for a trade. It contains the trade ID of the trade to which the settlement instructions belong. It will only be downloaded if settlement instructions have been requested.

4.2 Workflow

As trades within the FXtrades system undergo changes in response to user actions, those changes are communicated via FXintegrate On-Demand through the events described above. The following workflow rules apply to all events:

- NEWT, AMND, ROLL and ALOC are beginning events for trades. Each of these events will contain a new, unique trade ID. Those that are the result of changes to other trades (AMND, ROLL, ALOC) will contain references to the original trades.
- AMND starts a new trade and points back to its originating trade. This is a 1 to 1 relationship.
- ROLL and ALOC start new trades and point back to an originating trade. There may be many ROLLs or ALOCs for a single originating trade. Hence these are 1 to many events.
- An ALOC trade can have NO further events.
- CANC and CNDF end a trade's life. Such a trade cannot be amended, rolled or allocated.
- UDFA events can occur at any time in a trade's history and are not managed with a new trade ID.
- A CNDF event will be received for any trade that is cancelled, rolled, amended or allocated before its defined fields have been set. This is to ensure that some record of the original trade is sent even though it may contain no defined fields.
- NEWT, AMND, ROLL events will not be downloaded until any defined fields attached to the referenced trade have been set. This ensures that data required by downstream systems is not missing when a trade is downloaded.
- A trade cannot be allocated until its defined fields have been set.

4.3 Examples

The following examples illustrate the sequence of events that will occur for a given set of actions on a particular trade.

New Trade
NEWT

Allocated Trade
NEWT
ALOC
ALOC

Rolled Trade
NEWT
ROLL
ROLL
ROLL

Cancelled Trade
NEWT
CANC

Amended Trade
NEWT
AMND

New Trade with User Defined fields (UDF's), cancelled before UDF's are set
CNDF

New Trade with UDF's, rolled before UDF's are set
CNDF
ROLL
ROLL

5. Requirements

5.1 Account Creation

Prior to using the FXintegrate Real-Time service, the appropriate member account must first be created. Please contact Currenex Member Services to request an HTTPS account. Upon account creation, the member will receive an HTTPS user ID and account PIN. The user ID and account PIN will both be used during the digital certificate request process, described below.

5.2 Server Certificate

For FXintegrate Real-Time service to connect to a customer, the customer must present a valid server certificate that is issued by the commercial Certification Authority (CA).