Managed file transfer for SOA: A complete solution using WebSphere DataPower, WebSphere MQ, and WebSphere Service Registry and Repository

Stefano Marinoni
Luca Amato

01 July 2009

WebSphere MQ File Transfer Edition provides an enterprise-class platform for managed file transfer operations. This article shows you how to use File Transfer Edition to implement a managed file transfer system that enables you to issue file transfer commands simply by placing messages onto a queue and invoking a Web service.

Introduction

Different organizations or different departments in the same organization often need a reliable and secure method of enterprise-scale file transfer, ideally, one that lets applications or administrators issue file transfer requests automatically simply by invoking a Web Service. For example, consider a sales portal that needs to automatically generate and transfer files representing customer purchase invoices. Two IT systems, Source (S) and Destination (D) in different organizations, need an effective and reliable method of managed file transfer. This article shows you how to use IBM® WebSphere® MQ File Transfer Edition (hereafter called FTE) to implement an enterprise-scale file transfer solution. The solution also leverages WebSphere DataPower® SOA Appliances (hereafter called DataPower), and WebSphere Service Registry and Repository (hereafter called Service Registry), and it gives you a variety of ways to automate and control file transfers between IT systems.

FTE integrates with and runs over an existing WebSphere MQ network to add managed file transfer functionality. System administrators or applications can then issue messages containing file transfer directives for execution by FTE. Formatting messages correctly requires knowledge of the WebSphere MQ network, including agent names and file system structures. This article shows you how applications can issue file transfer commands simply by performing a Web service call to an ESB component deployed on DataPower, which is deployed as a bridge that listens for SOAP calls and subsequently populates the FTE command queues. DataPower supports inbound and outbound transactions over a variety of protocols -- the FTE solution in this article uses SOAP over HTTP(S) to raw XML over WebSphere MQ. The article also shows you how integration with Service Registry can provide a real-time view of the WebSphere MQ network as it changes over time, by automatically discovering agent names and file system structures that are regularly published into Service Registry, where they can be queried and used at runtime by the applications issuing the file transfer commands. From an architectural perspective, the FTE layer is constructed like any other WebSphere MQ application: it connects to queue managers to put messages on or get messages from queues. This article first describes FTE architecture to show how its components collaborate to implement reliable file transfer.

Acronyms used in this article

FTE	File Transfer Edition
XML	eXtensible Markup Language
SOAP	Single Object Access Protocol
S	Source
D	Destination

Prerequisites

In order to follow the examples in this article, you should have the following software installed in your test environment:

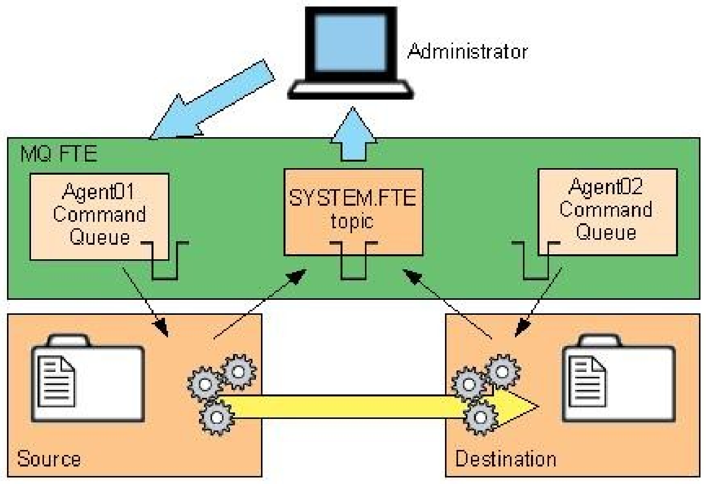
- WebSphere MQ FTE
- A WebSphere DataPower appliance with an application domain configured by importing the Dp-Domain.zip configuration file, which you can download at the bottom of the article.

FTE component architecture

FTE comes in different versions, depending on weather it will connect to a queue manager in bindings (locally), or in client mode. In addition to the functional components, each version also includes installation media containing remote tools and documentation packages. At configuration time, the administrator is asked to supply details of three queue managers: the Agent Queue Manager, Command Queue Manager, and Coordination Queue Manager. FTE is a single long-running daemon process called an agent, with tooling to let you submit commands to the agent and inquire on its status.

Figure 1 below shows an installation of an FTE instance without the queue managers included. Within the green rectangle are the basic components (agents and queues) that are the core of an FTE system. Also shown is an administration console that represents a remote system that can populate the agent's queue by issuing any kind of file transfer command, and two different file systems that represent the source and destination (target) system of a file transfer operation. In the green rectangle is a sender agent Agent 01 and a receiver agent Agent02, each with its own command queue. File transfers are managed by users or administrators by placing messages onto the sending agent's command queue, either directly or with the supplied tooling. A file transfer can be initiated via the command line, the Eclipse GUI, or by placing a properly formatted XML message on a queue. The agent processes respond to command messages, move files through FTE, and then publish activity logs onto the SYSTEM.FTE topic. Users, administrators, and applications that want log information can subscribe to the status updates on the SYSTEM.FTE topic.

Figure 1. Main WebSphere MQ FTE components



Deployment topologies

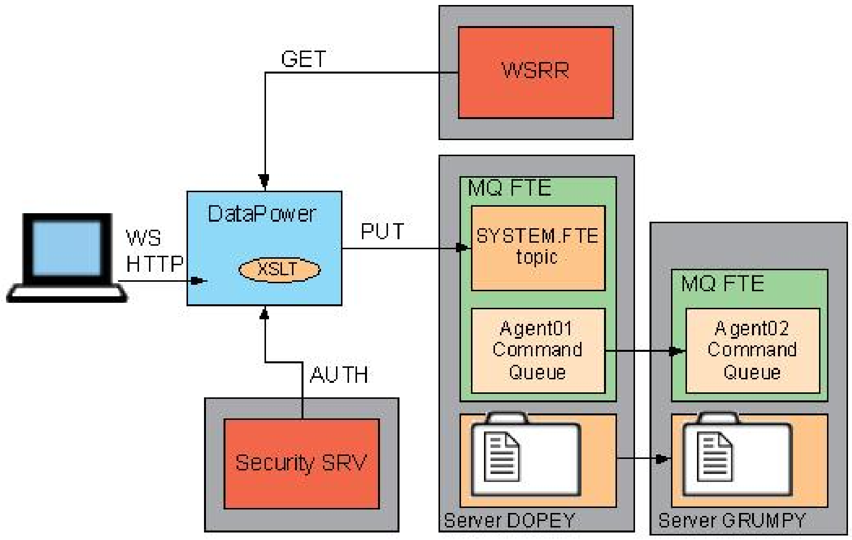
Where are the FTE components described above -- agents and relative queues -- normally deployed? Generally, each FTE component requires access to a queue manager. The agent and its queue manager may be hosted on the same server, or the agent may use a WebSphere MQ client channel to communicate with a remote queue manager. In the simplest case, the users and agents can all connect to the same queue manager. At the other extreme, the agents and users may be distributed across the network, with WebSphere MQ routing commands and data traffic between them. When discussing topologies, it is necessary to distinguish the different nodes based on their role.

Solution overview

The previous section gave an overview of FTE components to provide a consistent way to refer to them, independent of any particular platform or deployment configuration. This logical overview can now be linked to a specific FTE instance, to which additional components are added to create a complete managed file transfer solution.

Figure 2 below shows a runtime instance of the proposed FTE solution. A client entity requests a file transfer to be executed by invoking a web service. Then an ESB component (a DataPower Appliance) acts to map the incoming SOAP request into an XML file transfer request that is posted on an agent's queue, analogous to an administrator that issues a request for a file transfer operation. Instead of using the normal tooling, a Web Service invocation is used. Once the message is physically put on the Agent Command Queue, the file transfer is executed by an instance of FTE. In order for the ESB to build consistent file transfer commands, it must have knowledge of the MQ network, including agent and queue names. Service Registry enables the ESB to constantly get a real-time view of the MQ agents, and queues that might change over time.

Figure 2. Operational model



The FTE network topology is made up of two physical servers, DOPEY and GRUMPY. Each server with its own queue manager deploying agent and queues. Here is the environment for each of the two servers that are used to deploy an FTE instance:

DOPEY server

- Platform: Windows 2003 Server
- Queue Manager Name: QM_AgentHost1
- Agent Name: Agent01
- Hostname Listener: Win2k3Base on Port 1415
- Description: Windows® server used as a source for the file transfer demonstration

GRUMPY server

- Platform: Linux
- Queue Manager Name: QM_AgentHost2
- Agent Name: Agent02
- Hostname Listener: ESBdemo2 on Port 1414
- Description: Linux® server used as a destination for the file transfer demonstration

Even though these two agents use different queue managers running on different platforms, the messages produced when starting the agents are the same. Regardless or where the queue manager resides, you can send and receive files from any FTE agent to any other. For simplicity, the Service Registry instance in this solution contains only static metadata describing the FTE agent and queue name. A new Service Registry SupportPac enables it for automatic discovery of metadata relating to any FTE instance.

To summarize the main solution components:

Client
The entity that issues the file transfer request. The communication protocol between client and application server is based on HTTP.

DataPower
Front-end listener. The service is exposed with a Web services interface, and DataPower maps the client request into an FTE executable command. DataPower collaborates with Service Registry to discover the MQ agent's location and then prepare an XML message for the Command queue. Once the XML message is built, DataPower bridges the request using a different transport protocol than the incoming HTTP.

Service Registry
Discovers the MQ agent's location and maintains the information to make it available to DataPower during runtime.

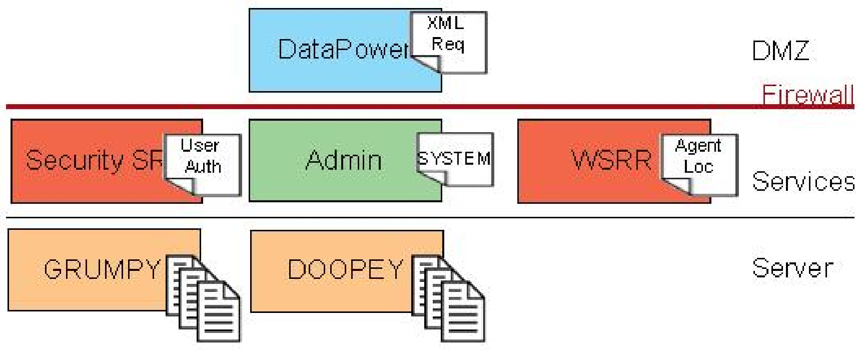
Security server
A user registry and directory that collaborates with DataPower to identify the requester client. Access management functions (authentication) can be enforced.

Server agents
The two FTE server agents (DOPEY and GRUMPY) are connected in an isolated network, and they are "on call: to perform the file transfer.

Technical benefits

Here are the architectural "zones" of the solution, including the logical components located on each layer of the software stack.

Figure 3. Software layer stack



DMZ (demilitarized zone)
A DataPower-managed layer where the service interface is exposed to the requesters. In other words, DataPower is the front end.

Services zone
Deploys all infrastructure services such as security, access manager, and FTE command queue, and manages all information required for service management.

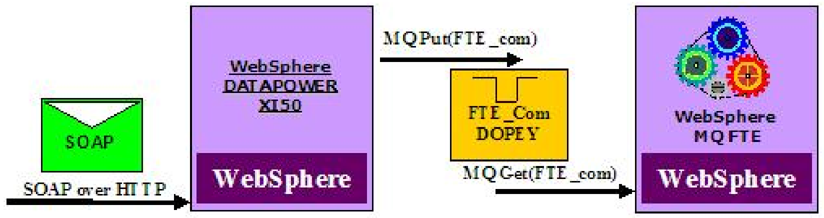
Back-end zone
The actual enterprise server farm, with the FTE agent running. In other words, this zone includes the file repositories. The back-end zone is the most critical one from a security point of view and is located in the internal security network zone.

The back-end system -- actually the entire FTE instance, as well as the source and destination file systems -- remain totally isolated and protected. Therefore the Web service requests issued by the client are managed directly by an ESB. Using DataPower as an ESB enables you to leverage capabilities that directly reflect gateway peculiarities such as security enforcement, message enrichment, and mapping. DataPower becomes the single point of entrance to the enterprise system, and is able to collaborate with all components in the Services zone. DataPower thus enables reliable file transfer as a service.

Configuration and implementation

Assume there is a file stored on a source file system named c:\file.txt that needs to be copied to a destination file system with the same name identifier on the path /tmp/file.txt. This basic scenario simply involves copying a file from DOPEY to GRUMPY using FTE. This section shows you how to configure an XML firewall on DataPower to listen for SOAP requests that will be mapped to basic XML FTE commands. Figure 4 shows the flow that must be implemented on DataPower to construct XML messages to be posted on the FTE command queue of the source agent:

Figure 4. Data elaboration flow for message mapping with DataPower



There is one XML firewall service (XML-FW) configured as a loopback proxy (Figure 4 only shows the request flow). The proxy deploys a basic policy that consists of a series of steps. It first calls Service Registry through an XSLT statement that fetches an XML specification containing the names of the FTE queue managers and agents stored on Service Registry. Once all the information needed to build an XML file transfer request is gathered, the message is mapped. Finally, the XML is posted on the command queue of the source agent (DOPEY). From here on, FTE executes the directives contained into the XML command issued by DataPower.

Message mapping sample

Here is the actual mapping from the SOAP input to the XML format that the FTE destination queue manager expects to receive to issue the file transfer commands:

Listing 1. Sample of the original SOAP message to be mapped into XML

```
<soapenv:Envelope
 xmlns:fte="http://fte.pot.ibm.com"
 xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:wsse=
 "http://oasis-200401-wss-wssecurity-secext-1.0.xsd"
 xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:dp="http://www.datapower.com/schemas/management">
 <soapenv:Header>
  <wsse:Security>
   <wsse:UsernameToken>
   <wsse:Username>david</wsse:Username>
   <wsse:Password>foobar</wsse:Password>
   </wsse:UsernameToken>
  </wsse:Security>
 </soapenv:Header>
 <soapenv:Body>
  <fte:copy>
   <sourceFilePath>c:\file.txt</sourceFilePath>
   <destinationFilePath>/tmp/file.txt</destinationFilePath>
  </fte:copy>
 </soapenv:Body>
</soapenv:Envelope>
```

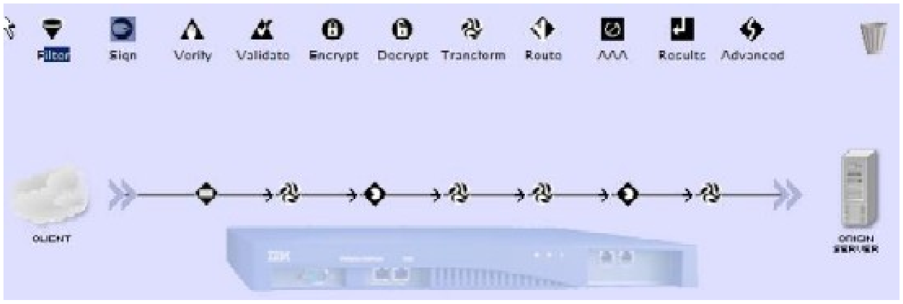
Listing 2. Sample of the mapped XML file containing FTE commands to be sent to an MQ agent for execution

```
<request
 xmlns:fte="http://fte.pot.ibm.com"
 xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:dp="http://www.datapower.com/schemas/management">
 <managedTransfer>
  <originator>
   <hostName>NASA_labs</hostName>
   <userID>SteLuke</userID>
  </originator>
   <sourceAgent agent="DOPEY" QMgr="QM_AGENTHOST1"/>
   <destinationAgent agent="GRUMPY" QMgr="QM_AGENTHOST2"/>
   <transferSet priority="0">
   <item mode="binary" checksumMethod="MD5">
   <source recursive="false" disposition="leave">
    <file>c:\file.txt</file>
   </source>
   <destination type="file" exist="overwrite">
    <file>/tmp/file.txt</file>
   </destination>
   </item>
   </transferSet>
 </managedTransfer>
</request>>
```

Flow design

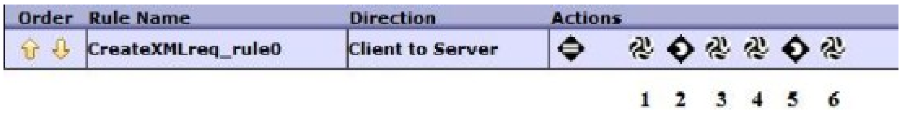
This section illustrates the graphical aspect of the flow through the XML firewall described above, focusing on the request flow from client to server:

Figure 5. Request flow of the DataPower policy



The flow consists of a single rule named CreateXMLreq_rule0, and its steps are described below:

Figure 6. DataPower rule to perform the mapping



Order	Rule Name	Direction	Actions
	CreateXMLreq_rule0	Client to Server	

1. **Fetch from Service Registry** -- Before the rule execution begins, the SOAP message is matched via an HTTP Front Side Handler listening for SOAP requests on DataPower Port 1011. Subsequently, the initial action is an XSLT transform that calls Service Registry to fetch an XML artifact file containing the definition of the source and destination queue managers, and the agents of the FTE instance, using an XSLT extension function of DataPower that makes a SOAP call to Service Registry:

   Listing 3. Stylesheet to call Service Registry to fetch an XML artifact: (see FetchFromWSRR-agentsDescription.xsl)

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:dp="http://www.datapower.com/extensions"
exclude-result-prefixes="dp" extension-element-prefixes="dp">
<xsl:output method="xml"/>
   <!-- Fetching the WS-Policy file directly from Service Registry.
    Use dp:soap-call() to send a message to Service Registry
    and save the response in a result variable. -->
 <xsl:variable name="request-payload">
  <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soap:Header/>
   <soap:Body>
    <p244:retrieve xmlns:p244="http://fte.pot.ibm.com">
    <bsrURI>b5030db5-da87-47ab.be7b.828d3c827b06</bsrURI>
    </p244:retrieve>
   </soap:Body>
  </soap:Envelope>
 </xsl:variable>
 <xsl:template match="/*">
  <!-- Ccontent type = XML, Action = retrieve -->
  <xsl:variable name="httpHeaders">
   <header name="Content-Type">text/xml</header>
   <header name="SOAPAction">retrieve</header>
  </xsl:variable>
  <xsl:variable name="url">
   <wsrrURL name="urlvalue">
   http://wsrrserver:9080/WSRRCoreSDO/services/WSRRCoreSDOPort
   </wsrrURL>
  </xsl:variable>
  <xsl:variable name="resultSOAPcall"
  select="dp:soap-call($url,$request-payload, '',0,'',$httpHeaders)"/>
  <xsl:copy-of select="$resultSOAPcall"/>
  <!-- Setting up the WS-policy value retrieved -->
  <dp:set-variable name="'var://context/myContext/MyFetchedInformation'"
  value="$resultSOAPcall"/>
 </xsl:template>
</xsl:stylesheet>
```

2. **Extract using X-path** -- A basic set variable action. Once Service Registry responds, it returns an XML file containing an attribute named content whose value is the actual artifact

queried from Service Registry. This attribute is a base-64 encoded string containing both source and destination agent names with relative queue managers. Assign the DataPower variable var://context/myContext/MyFetchedInformation the value of this attribute.

3. **Decode base-64 encoded string** -- XSLT transform step that decodes the cipher value of the variable var://context/myContext/MyFetchedInformation. Next, the agent and queue managers names will be there to be used by DataPower to construct the XML command:

   ### Listing 4. Stylesheet to decode the artifact fetched from Service Registry (see DecodeFetched.xsl)

```
<?xml version="1.0"?>
 <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:dp="http://www.datapower.com/extensions"
 exclude-result-prefixes="dp" extension-element-prefixes="dp">
 <xsl:output method="xml"/>
 <!-- Decoding from base64 to plain text string. Will it ever work? hmmmmmm -->
 <xsl:template match="/*">
 <xsl:variable name="EncodedPolicy"
 select="dp:variable('var://context/myContext/MyFetchedInformation')"/>
 <xsl:variable name="DecodedPolicy"
  select="dp:decode($EncodedPolicy,'base-64')"/>
 <xsl:copy-of select="$DecodedPolicy"/>
 <xsl:variable name="DecodedPolicy" select="dp:parse($DecodedPolicy)"/>
 <xsl:copy-of select="$DecodedPolicy"/>
 <dp:set-variable name="'var://context/myContext/MyFetchedInformation'"
 value="$DecodedPolicy"/>
 <!-- Setting up the WS-policy value retrieved -->
 </xsl:template>
</xsl:stylesheet>
```

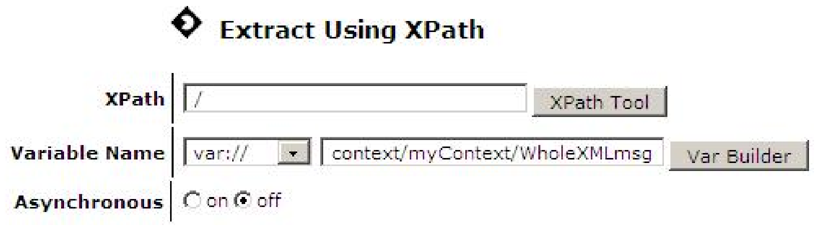
4. **Create XML message to issue file transfer** -- The current action is an xslt transform step that builds an XML request to transfer the file from source to destination file system. The XML message to be posed in the FTE command queue is firstly created and put temporarily into the output context of the current step; subsequently is stored into a DataPower variable named var://context/myContext/WholeXMLmsg:

   ### Listing 5. Stylesheet to build a proper XML message to be sent for the FTE command queue. (see MakeAnXMLCommand.xsl)

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:fte="http://fte.pot.ibm.com"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:dp="http://www.datapower.com/extensions"
exclude-result-prefixes="dp" extension-element-prefixes="dp"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
<xsl:output method="xml"/>
<xsl:template match="/">
 <xsl:variable name="SourceAgent"
 select="dp:variable('var://context/myContext/MyFetchedInformation')"/>
 <xsl:variable name="SourceAgent"
 select="string($SourceAgent/agents/sourceAgent/@agent)"/>
 <dp:set-variable
 name="'var://context/myContext/sourceAgent'" value="$SourceAgent"/>
 <xsl:variable name="SourceQmgr"
 select="dp:variable('var://context/myContext/MyFetchedInformation')"/>
 <xsl:variable name="SourceQmgr"
 select="string($SourceQmgr/agents/sourceAgent/@QMgr)"/>
 <dp:set-variable name="'var://context/myContext/SourceQmgr'"
 value="$SourceQmgr"/>
 <xsl:variable name="destinationAgent"
 select="dp:variable('var://context/myContext/MyFetchedInformation')"/>
 <xsl:variable name="destinationAgent"
 select="string($destinationAgent/agents/destinationAgent/@agent)"/>
 <dp:set-variable name="'var://context/myContext/destinationAgent'"
 value="$destinationAgent"/>
 <xsl:variable name="destinationQmgr"
 select="dp:variable('var://context/myContext/MyFetchedInformation')"/>
 <xsl:variable name="destinationQmgr"
 select="string($destinationQmgr/agents/destinationAgent/@QMgr)"/>
 <dp:set-variable name="'var://context/myContext/destinationQmgr'"
 value="$destinationQmgr"/>
 <request>
  <managedTransfer>
   <originator>
    <hostName>NASA_labs</hostName>
    <userID>SteLuke</userID>
   </originator>
   <xsl:element name="sourceAgent">
    <xsl:attribute name="agent">
    <xsl:value-of select="$SourceAgent"/></xsl:attribute>
   <xsl:attribute name="QMgr">
    <xsl:value-of select="$SourceQmgr"/></xsl:attribute>
    </xsl:element>
   <xsl:element name="destinationAgent">
    <xsl:attribute name="agent">
    <xsl:value-of select="$destinationAgent"/>
    </xsl:attribute>
   <xsl:attribute name="QMgr">
    <xsl:value-of select="$destinationQmgr"/>
    </xsl:attribute>
   </xsl:element>
   <transferSet priority="0">
   <item mode="binary" checksumMethod="MD5">
   <source recursive="false"
   disposition="leave">
   <file>
   <xsl:value-of
   select="./soapenv:Envelope/
   soapenv:Body/fte:copy/sourceFilePath"/>
   </file>
   </source>
   <destination type="file"
   exist="overwrite">
   <file>
   <xsl:value-of
   select="./soapenv:Envelope/
   soapenv:Body/fte:copy/destinationFilePath"/>
   </file>
   </destination>
   </item>
   </transferSet>
  </managedTransfer>
 </request>
</xsl:template>
</xsl:stylesheet>
```

5. **Extract using X-path** -- A basic set variable action. Assign the DataPower variable var://context/myContext/WholeXMLmsg the value of the context output of the previous step. Now you just have to post that message to the FTE command queue.

Figure 7. Configuration details for the set Var action



Extract Using XPath

XPath: /

Variable Name: var:// context/myContext/WholeXMLmsg

Asynchronous: on [✓ off]

6. **Post XML message on FTE command queue** -- The final step is to read from var://context/myContext/WholeXMLmsg to put the XML message on the FTE command queue of the source agent, namely SYSTEM.FTE.COMMAND.DOPEY:

   Listing 6. Making an MQ put from DataPower Flow. (see MakeRealCallandBuidReply.xsl)

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:dp="http://www.datapower.com/extensions"
exclude-result-prefixes="dp" extension-element-prefixes="dp">
 <xsl:output method="xml"/>
 <!-- We have our own request stored in
  dp:variable('var://context/myContext/WholeXMLmsg') -->
 <xsl:template match="/*">
  <xsl:variable name="request-payload"
  select="dp:variable('var://context/myContext/WholeXMLmsg')"/>
  <!-- Posting a file transfer command request on proper queue -->
  <xsl:variable name="ResultMQcall">
   <dp:url-open
   target="dpmq://MQ-FTE/?RequestQueue=SYSTEM.FTE.COMMAND.DOPEY"
   response="binaryNode" resolve-mode="xml" data-type="xml">
   <xsl:copy-of
   select="dp:variable('var://context/myContext/WholeXMLmsg')"/>
   </dp:url-open>
  </xsl:variable>
  <!-- Building the basic HttpResponse to
   a symbolic XML message to the caller -->
  <xsl:variable name="XMLresponse">
   an XML message requesting a
   filetransfer has been SUCCESFULLY issued to MQ-fte AGENT
  </xsl:variable>
  <xsl:copy-of select="$XMLresponse"/>
  <!-- Initializing a local variable -->
  <dp:set-variable name="'var://context/myContext/MQcall'"
   value="$ResultMQcall"/>
 </xsl:template>
</xsl:stylesheet>
```

You have finished the exercise by getting the final XML message that will issue a file transfer command to the FTE system.

Conclusion

Although FTE provides a wide set of tools for the administrator to issue file transfer commands, most of these are intended for interacting directly with the user. Therefore they are often GUI-based or require the use of a specific MQ API. When you need an enterprise application to issue file transfer commands directly to the FTE system, you need a platform-independent interface that does not require the use of any particular API, and FTE accepting commands through a Web service interface meets this requirement. This article showed you how to enable the FTE system to accept file transfer requests on a Web service listener, showed you how to integrate this solution with Service Registry to obtain a real-time view of the FTE network as it changes over time.

Acknowledgements

The authors would like to thank Pierangelo Giusta of IBM Italy, who collaborated with the authors on integrating the WebSphere products described in this article.

Downloadable resources

Description	Name	Size
A set of stylesheets for the DataPower flow	Artifacts_xslt.zip	402 KB