Forte Fusion Backbone Integration Guide

Release 2.1 of Forte Fusion™

Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, CA 94303 U.S.A. 1-800-786-7638

Part No. 806-7137-01
November 2000, Revision A

Contents

Preface

1 Introduction

3 Synchronous Service Provider Use Case

4 Asynchronous Service Provider Use Case

5 Service Requestor Authentication Use Case

6 Service Provider Authentication Use Case

7 Proxy Recovery Use Case

8 Application Recovery Use Case

9 Independent Proxy Use Case

10 Independent Proxy Authentication Use Case

A Transforming Proxy Documents

B Proxy Document Element Hierarchies

Preface

The Forte Fusion Backbone is an enterprise application integration product that enables communication between disparate applications. Forte Fusion enables platform-independent automation and control of business processes through the use of a Fusion process engine. Fusion also can integrate applications directly, without coordinating activities through a process engine.

The *Forte Fusion Backbone Integration Guide* describes how to use Fusion to integrate applications. Integrated applications exchange XML (Extensible Markup Language) messages among themselves and Fusion application proxies. The manual explains how to write XSL (Extensible Stylesheet Language) stylesheets, using XSLT (Extensible Stylesheet Language Transformations) to transform XML documents. In this way, an application or an application proxy can send documents that the recipient can understand and act upon. The information is presented in a series of use cases that represent the most common scenarios in a Fusion enterprise application.

The manual also provides introductory information about XSLT and the basic transformations used with Fusion. Finally, the manual helps you perform tasks related to integrating applications, including proxy configuration and application design considerations.

This book is intended primarily for users who write XSL stylesheets, and secondarily for application developers and Fusion system administrators. It assumes familiarity with Forte Fusion (including process management concepts and Forte system management concepts, as well as the HTTP protocol, XML, and XSL/XSLT. You also should be familiar with the Fusion proxy documents and their XML vocabulary and structure. For information about proxy document XML, see the Fusion Backbone online Help. For a list of related sources of information, see “Documentation” on page 15.

About Forte Fusion

Forte Fusion is a suite of business integration tools for integrating and coordinating heterogeneous applications. The tools and software components provided with Forte Fusion let you integrate newly developed applications, legacy applications, and off-the-shelf packages into business processes that are automated and controlled by a process engine.

A Fusion system is a set of tools and software modules installed on top of a compatible version of the Forte Application Environment. It is composed of two subsystems, a process management system and an XML-based backbone system.

Fusion Process Management System The Fusion Process Management System (formerly known as Conductor) provides a set of tools and software modules that support the development, execution, and management of business processes. The heart of this system is the Fusion Process Engine, which controls and manages business processes from beginning to end, coordinating the work of the different resources or applications that participate in the processes.

Forte Fusion customers use the Fusion Process Management System to:

- develop process logic with the graphical process development workshops
- manage sessions and processes, and the engine itself, using the Fusion Console and other tools
- build applications, called process clients, that make direct API calls to the process engine, using the process client APIs (Forte 4GL, CORBA/IIOP, JavaBeans, ActiveX, or C++)

Backbone System The Fusion Backbone System provides a set of tools and software modules that use XML messaging over HTTP or JMS to simplify communication and coordination between applications. A Fusion backbone can support different styles of integration, but the backbone is always installed on top of the Fusion process engine runtime. The heart of a backbone system is a set of application proxies that perform message brokering and data transformation on behalf of applications. For business process support, proxies interact with the Fusion process engine on behalf of any applications that participate in a common business process. The main purpose of these interactions is to communicate the initiation and completion of work activities.

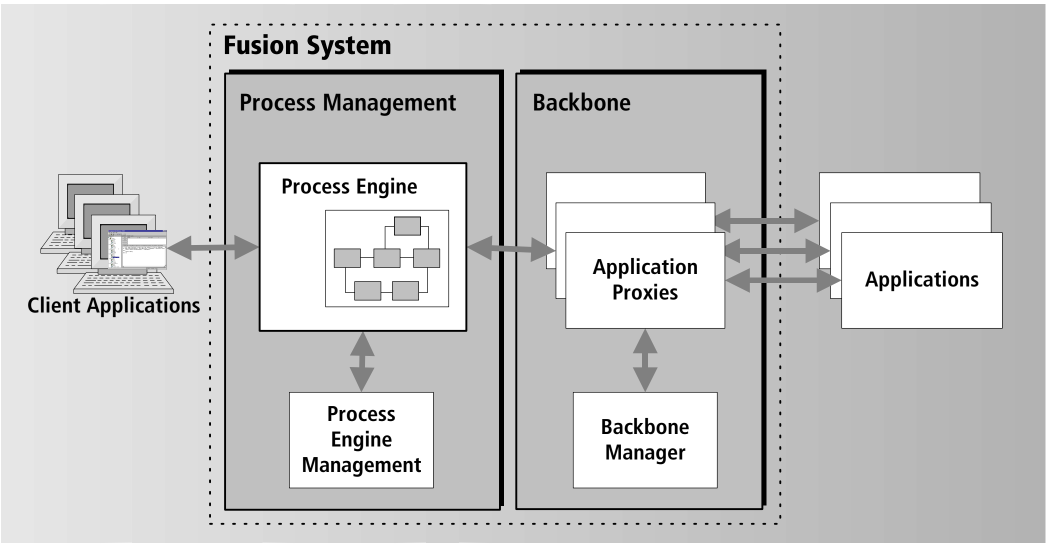


Figure 1 Fusion System and Subsystems

Forte provides adapters as well as an adapter toolkit to integrate packages or custom applications that lack a native XML/HTTP interface into a Fusion backbone.

Forte Fusion customers use the Fusion Backbone System primarily to:

- provide an XML/HTTP interface between proxies and applications
- configure application proxies to participate in a managed business process

The XML/XSL Workshops provided with Fusion facilitate the development, testing, debugging, storage, and management of sample XML documents and the XSL stylesheets used for message transformation between applications.

Organization of this Manual

This manual begins with an introduction to XSL and guidelines for writing XSL stylesheets, in general and in the context of a Fusion enterprise application. Most of the manual consists of use cases that provide procedures for enabling the exchange of XML messages in the most common scenarios you are likely to encounter. The manual also provides appendixes that provide information about the flow and structure of the XML documents that Fusion uses.

Briefly, the chapters of this manual are:

Chapter	Description
Chapter 1, "Introduction"	Provides an overview of the use cases presented in this manual, XSL stylesheets, the Fusion example application, and XSLT transformations
Chapter 2, "Service Requestor Use Case"	Describes how to enable a service requestor application to start a Fusion process
Chapter 3, "Synchronous Service Provider Use Case"	Describes how to enable a Fusion proxy to request that a partner application perform an activity; in this case, the proxy waits for a response from the application
Chapter 4, "Asynchronous Service Provider Use Case"	Describes a similar scenario to Chapter 3, except that the application responds to the proxy during a subsequent HTTP session
Chapter 5, "Service Requestor Authentication Use Case"	Describes how to enable a partner application to authenticate itself to a proxy when attempting to start a Fusion process
Chapter 6, "Service Provider Authentication Use Case"	Describes how to enable a proxy to authenticate itself to a service provider application when requesting that the application perform an activity
Chapter 7, "Proxy Recovery Use Case"	Describes how to reestablish the flow of message between a proxy and a partner application after the proxy has failed and recovered
Chapter 8, "Application Recovery Use Case"	Describes how to reestablish the flow of message between a proxy and a partner application after the application has failed and recovered
Chapter 9, "Independent Proxy Use Case"	Describes how to use an independent proxy to transmit XML documents between applications without being connected to a Fusion process engine
Chapter 10, "Independent Proxy Authentication Use Case"	Describes how to enable an application to authenticate itself to an independent proxy
Appendix A, "Transforming Proxy Documents"	Explains how messaging works within the proxy according to the proxy's function in a Fusion application system
Appendix B, "Proxy Document Element Hierarchies"	Diagrams illustrating the element hierarchy and vocabulary of Fusion proxy documents

Conventions

This manual uses standard Forte documentation conventions in specifying command syntax and in documenting Forte 4GL TOOL code.

Command Syntax Conventions

The specifications of command syntax in this manual use a “brackets and braces” format. The following table describes this format:

Format	Description
bold	Bold text is a reserved word; type the word exactly as shown.
italics	Italicized text is a generic term that represents a set of options or values. Substitute an appropriate clause or value where you see italic text.
UPPERCASE	Uppercase text represents a constant. Type uppercase text exactly as shown.
underline	Underlined text represents a default value.
vertical bars |	Vertical bars indicate a mutually exclusive choice between items. See braces and brackets, below.
braces { }	Braces indicate a required clause. When a list of items separated by vertical bars is enclosed in braces, you must enter one of the items from the list. Do not enter the braces or vertical bars.
brackets [ ]	Brackets indicate an optional clause. When a list of items separated by vertical bars is enclosed by brackets, you can either select one item from the list or ignore the entire clause. Do not enter the brackets or vertical bars.
ellipsis …	The item preceding an ellipsis may be repeated one or more times. When a clause in braces is followed by an ellipsis, you can use the clause one or more times. When a clause in brackets is followed by an ellipsis, you can use the clause zero or more times.

Forte 4GL TOOL Code Conventions

Where this manual includes documentation or examples of Forte 4GL TOOL code, the TOOL code conventions in the following table are used.

Format	Description
parentheses ( )	Parentheses are used in TOOL code to enclose a parameter list. Always include the parentheses with the parameter list.
comma ,	Commas are used in TOOL code to separate items in a parameter list. Always include the commas in the parameter list.
colon :	Colons are used in TOOL code to separate a name from a type, or to indicate a Forte name in a SQL statement. Always include the colon in the type declaration or statement.
semicolon ;	Semicolons are used in TOOL code to end a TOOL statement. Always type a semicolon at the end of a statement.

Fusion Example Programs

Forte provides a number of example applications that illustrate Fusion features.

Process Client Example Programs

There are five different APIs available to build a Fusion process client. Example application programs are provided for each API. Each API has its own example files in a subdirectory of FORTE_ROOT/install/examples/conductr/. The PDF file in the examples subdirectory explains how to install and run the example application.

The examples are described in the appendix of the *Forte Fusion Process Development Guide*.

Backbone Example Programs

There are three Forte Fusion Backbone example programs: one illustrating the use of the Forte 4GL TOOL with Fusion, another is an example written in ‘C’, and a third example illustrates JMS messaging.

The example programs are installed under FORTE_ROOT/install/examples/fusion.

The directory containing each example includes a readme file. The readme file contains the background information and configuration instructions. The Forte 4GL example is a complete system, while the ‘C’ example is an optional replacement for some parts of the Forte 4GL example.

Documentation

The Fusion online documentation includes the complete documentation set and a master index as PDF (Portable Document Format) files as well as online help. For details on viewing and searching these files, see "Viewing and Searching PDF Files" on page 17.

When you are using a Fusion development application, press the Help key or use the Help menu to display online help. The help files are also available at the following location in your Fusion distribution: FORTE_ROOT/userapp/forte/cl*n*/*.hlp (*n* indicates the release number).

When you are using a script utility, such as Conductor Script (Cscript) or Fusion Script (FNscript), type help from the script shell for a description of all commands, or help <*command*> for help on a specific command.

Forte Fusion Documentation Resources

The *Forte Fusion Installation Guide* explains installation options and how to install the Fusion product (both the Fusion Process Engine and the Fusion Backbone).

Other useful resources available in the Fusion product documentation directory are:

- the fndoc.pdf file which serves as a home page for the entire documentation set
- a master index for all Fusion PDF documentation
- a glossary of terms
- a list of resources for learning more about the underlying technologies

Forte Fusion Process Management

The complete documentation set for Forte Fusion Process Management consists of the following manuals and online help:

- *Forte Fusion Process Development Guide*. Explains how to create business process logic using the graphical process development workshops.
- *Forte Fusion Process Management System Guide*. Explains system management concepts and facilities, how to register process definitions, how to configure and manage the process engine, and other related tasks.
- *Forte Fusion Process Client Programming Guide*. If you are building new applications that interact directly with the process engine, this manual explains how to use one of the provided process client APIs for that purpose. Use in conjunction with the API reference in the online help.
- Online help. Provides complete API reference for the process client APIs as well as task Help on the workshops and the Fusion Console.

Forte Fusion Backbone

The documentation set for the Forte Fusion Backbone consists of the following manuals and online help:

- *Forte Fusion Backbone System Guide*. Explains the backbone architecture, proxy concepts and features, and how to configure backbones and proxies. It includes a reference for FNscript, the Fusion scripting language. This should be the first manual you read if you are integrating an application or an adapter into a Fusion backbone.
- *Forte Fusion Backbone Integration Guide*. Explains how to develop XSL stylesheets and perform other integration tasks so that appropriate XML message transformations can occur between applications and proxies. The manual is used in conjunction with the *Forte Fusion Backbone System Guide* and the Fusion Backbone online help.
- Fusion Backbone online help. Explains proxy document XML, how to use the XML/XSL workshops to create, debug, and manage XML documents and XSL stylesheets, how XSLT and standard XML parsers are supported in Fusion, and provides a complete reference for the HTTPSupport (formerly the HTTP-DC) API. The Forte HTTPSupport API enables the development of standard HTTP communication services for transporting and managing HTML and XML messages.
- *Forte Fusion Adapter Development Guide*. Explains how to use the Forte Fusion software development kit to build a custom Fusion adapter for HTTP services. Discusses the design and functioning of a Fusion adapter, and includes guidance on using the ‘C’ and Forte 4GL TOOL adapter SDKs. Use in conjunction with the Fusion Backbone online help, which explains the HTTPSupport API, as well as with the adapter readme files. The following table indicates the location of readme files that contain further information about the ‘C’ adapter functions and the ‘C’ and TOOL adapter example programs:

Contents	Location
Installing TOOL adapter example program	FORTE_ROOT/install/examples/fusion/toolcon/readme.htm
Installing ‘C’ adapter example programs	FORTE_ROOT/install/examples/fusion/ccon/readme.htm
Explanation of ‘C’ adapter functions and #defines	FORTE_ROOT/install/fusion/ccon/fnconect.htm

Forte Application Environment

Forte provides a comprehensive documentation set describing the libraries, languages, workshops, and utilities of the Forte Application Environment. For the complete Forte Release 3 documentation set, see the Forte documentation listed on the Forte CyberSupport page at *http://www.forte.com/support*.

Viewing and Searching PDF Files

You can view and search Fusion PDF files directly from the product CD-ROM, store them locally on your computer, or store them on a server for multiuser network access.

There are two ways you can look up information in the Fusion documentation set:

- view and search PDF files directly from the product CD-ROM

  The Fusion documentation set has been indexed with Acrobat Catalog. Use Acrobat Reader with Search to search for text strings across the book set and click hypertext links to display the specified content.

- look up index entries in the *Forte Fusion Master Index,* included on the product CD-ROM

  The master index also helps you find content across the full documentation set. It is a composite of all Fusion book indexes and is intended to be displayed online or printed to your local printer. It does not provide hypertext links to entries as the individual book indexes do.

Note You need Acrobat Reader 4.0+ to view and print the files. Acrobat Reader with Search is recommended and is available as a free download from http://www.adobe.com. If you do not use Acrobat Reader with Search, you can only view and print files; you cannot search across the collection of files.

▶ **To copy the documentation to a client or server:**

1. Copy the fortedoc directory and its contents from the CD-ROM to the client or server hard disk.

   You can specify any convenient location for the fortedoc directory; the location is not dependent on the Forte distribution.

2. Set up a directory structure that keeps the fndoc.pdf and the fusion directory in the same relative location.

   The directory structure must be preserved to use the Acrobat search feature.

Note To uninstall the documentation, delete the fortedoc directory.

▶ **To view and search the documentation:**

1. Open the file fndoc.pdf, located in the fortedoc directory.
2. Click the **Search** button at the bottom of the page or select **Edit** > **Search** > **Query**.
3. Enter the word or text string you are looking for in the Find Results Containing Text field of the Adobe Acrobat Search dialog box, and click **Search**.

   A Search Results window displays the documents that contain the desired text. If more than one document from the collection contains the desired text, they are ranked for relevancy.

Note For details on how to expand or limit a search query using wild-card characters and operators, see the Adobe Acrobat Help.

4. Click the document title with the highest relevance (usually the first one in the list or with a solid-filled icon) to display the document.

   All occurrences of the word or phrase on a page are highlighted.

5 Click the buttons on the Acrobat Reader toolbar or use shortcut keys to navigate through the search results, as shown in the following table:

Toolbar Button	Keyboard Command
Next Highlight	Ctrl+]
Previous Highlight	Ctrl+[
Next Document	Ctrl+Shift+]

6 To return to the fndoc.pdf file, click the Homepage bookmark at the top of the bookmarks list.

7 To revisit the query results, click the **Results** button at the bottom of the fndoc.pdf home page or select **Edit** > **Search** > **Results**.

Chapter 1

Introduction

This chapter provides an overview of the tasks and concepts involved in using Extensible Markup Language (XML) and Extensible Stylesheet Language (XSL) to integrate applications into a Fusion application. The chapter also discusses how to use this manual and related tools for this purpose.

This chapter covers the following topics:

- the Fusion use cases presented in the following chapters
- the sample application included with your Forte installation
- XSL stylesheets and XSL transformations
- general guidelines for writing XSL stylesheets
- handling common Fusion integration tasks in your XSL stylesheets

Working with the Fusion Use Cases

Each of the following chapters of this manual describes a Fusion use case—a scenario that represents a typical interaction between an application and the Fusion backbone. Taken together, these cases present most of the typical situations that require XSL stylesheets while using Fusion.

The following table lists the use cases presented in this manual:

Chapter	Use Case
Chapter 2, "Service Requestor Use Case"	service requestor starts process
Chapter 3, "Synchronous Service Provider Use Case"	service provider with synchronous processing
Chapter 4, "Asynchronous Service Provider Use Case"	service provider with asynchronous processing
Chapter 5, "Service Requestor Authentication Use Case"	service requestor with FusionXML authentication
Chapter 6, "Service Provider Authentication Use Case"	service provider with FusionXML authentication
Chapter 7, "Proxy Recovery Use Case"	proxy fails and recovers
Chapter 8, "Application Recovery Use Case"	application fails and recovers
Chapter 9, "Independent Proxy Use Case"	applications exchange XML documents through independent proxy without a Fusion process engine
Chapter 10, "Independent Proxy Authentication Use Case"	independent proxy requires authentication from application (or must provide authentication to application)

For each case, the chapter provides a description of the case and procedures for enabling its successful completion. These procedures describe how to write any required inbound or outbound stylesheets, as well as other actions you must take, for example, configuring a proxy with FNscript commands or defining Fusion process attributes.

The following table describes the information provided about each use case:

Use Case Information	Description
description	a brief description of the use case
expected outcome	the actions that constitute the successful completion of the case, for example, a Fusion process is started or an application performs an activity
actors	the Fusion components and applications that take part in the case
proxy document flow	the flow of proxy documents required for the case
required stylesheets	the XSL stylesheets you write to enable the case to be completed successfully
other integrator tasks	any additional actions you must perform, for example, configuring a proxy

For a general overview of the Fusion processing involved in the service requestor and service provider use cases, see Appendix A, "Transforming Proxy Documents."

Using the Fusion Example Application

Your Fusion installation includes an example application based on a Forte 4GL TOOL adapter. The example represents an electronic customer order processing system, with activities such as placing an order, verifying credit, and shipping the order.

You can find the example application in the following directory of your Fusion installation:

```
//FORTE_ROOT/install/examples/Fusion/toolcon
```

For information about installing and running the application, see the *readme.htm* file included with the application.

Wherever possible, the use cases in this manual follow the example application. When the application does not cover the scenario presented by a use case, the case conceptually extends the example. Some examples from the application have been modified slightly for the sake of brevity or clarity.

Example Application Stylesheets

The example application uses two XSL stylesheets:

- *orderin.xsl*, an inbound stylesheet that transforms application documents into command documents
- *orderout.xsl*, an outbound stylesheet that transforms state documents into command documents and application documents

These stylesheets have been written to provide examples of a wide range of transformations required in integrating applications with Fusion. They are not meant to be taken as guides to writing stylesheets for an actual order processing system.

Also, in the example application, all proxies share one set of stylesheets. This implementation works in the example, because each of the activities uses the same XML vocabulary and structure.

In actual practice, each activity might be a different type of application, for example a legacy mainframe application or a third-party packaged application, with its own XML vocabulary and structure. In such a real-world context, each proxy probably would have its own set of stylesheets.

Note The stylesheet *nopein.xsl* is included as an example of an inbound stylesheet for use with independent proxies.

Understanding XSL Stylesheets and XSL Transformations

An XSL stylesheet is an XML document that contains templates for transforming a source document into a results document. In a Fusion enterprise application, each application proxy has two associated stylesheets:

- an inbound stylesheet that transforms an application document into a command document

  A command document instructs the proxy's command processor to send instructions to the Fusion process engine, to send messages to applications, or both. The inbound stylesheet contains the rules that specify the commands and messages to send.

- an outbound stylesheet that transforms a state document into command documents and application documents

  A state document is produced by a proxy and is based on information from the process engine about the state of a process or activity. The outbound stylesheet contains rules that specify how to generate a command document with further commands to the engine, messages to applications, or both.

  Figure 2 shows how a proxy applies an inbound stylesheet to an application document.

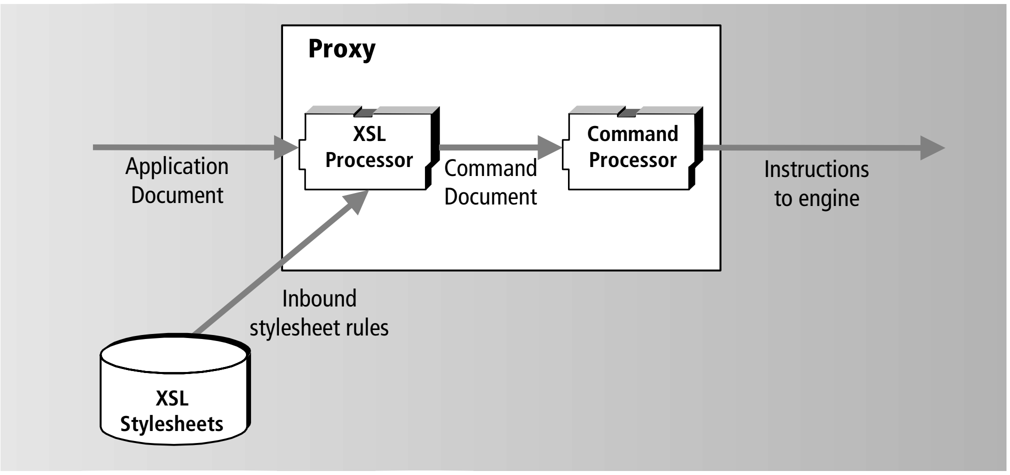


Figure 2 *Applying an Inbound Stylesheet*

Note The flow is reversed for outbound stylesheets, that is, the stylesheet is applied to a state document, which is transformed into a command document that can produce an application document.

A proxy can have one stylesheet that provides both inbound and outbound transformations, although this is generally not good practice.

The following sections provide:

- an overview of the kinds of transformations your XSL stylesheets should contain
- an introduction to the XML/XSL workshops that Fusion provides for developing, testing, and debugging stylesheets and for managing collections of stylesheets and sample XML documents

For more information about proxy documents, see the *Forte Fusion Backbone System Guide* and the Fusion Backbone online help.

Writing XSL Transformations

XSL stylesheets produce results documents by matching patterns in a source XML document. For each pattern that your stylesheet matches, you provide a template that describes how to render that pattern in the results document. To achieve this transformation, you include an xsl:template element with the following format:

```
<xsl:template match="SourcePattern">
  do something with the source document pattern that was matched...
</xsl:template>
```

The value of the "match" attribute is the pattern you want to match in the source document. This value can be as simple as an element name, or it can be a complicated pattern that matches particular values somewhere in the source document.

Between the start and end tags of the xsl:template element, you use other XSL elements to specify a template for rendering the item in the results document. The following example matches an element called MsgDoc in the source document and transforms it into an "FNCommand" element in a resulting command document:

Match the MsgDoc element
Create a new element
End tag

```
<xsl:template match="MsgDoc">
  <xsl:element name="FNCommand"/>
</xsl:template>
```

While this template is very simple, you can use XSLT to specify more complex templates. For example, the following template finds an element named Att in an application document and transforms it into an FNProcessAttribute element (that corresponds to a Fusion process attribute). The template also creates a Name attribute for the new element; the value of this attribute is the value of the AttName element in the source document:

Match the Att element
Create a new element
Create a new attribute
Copy AttName element value
End tag

```
<xsl:template match="Att">
  <xsl:element name="FNProcessAttribute">
    <xsl:attribute name="Name">
      <xsl:value-of select="AttName"/>
    </xsl:attribute>
  </xsl:element>
</xsl:template>
```

In addition to creating elements and attributes with different names, your templates can apply functions to recompute source values, copy sections of the source tree directly into the results tree, or specify more complex actions. Thus, your XSL stylesheets can provide a wide range of transformations that might be needed in the fulfillment of your business process.

Using Inbound and Outbound Stylesheets

A proxy is usually configured with two stylesheets:

- Outbound stylesheets specify the rules applying to state documents produced by the Fusion process engine.
- Inbound stylesheets specify the rules applying to documents sent by the application to the proxy.

The same stylesheets can apply in both directions, if appropriate.

Note An independent proxy—a proxy that is does not communicate with a process engine—requires only an inbound stylesheet.

The following sections provide a brief overview of the kinds of transformations that inbound and outbound stylesheets might contain. For examples of Fusion stylesheets, see the files *orderin.xsl* and *orderout.xsl* in the FORTE_ROOT/install/examples/fusion/toolcon directory. These files are part of the TOOL adapter example application.

Inbound Transformations

An inbound stylesheet provides templates for transforming an application document into a command document that tells the proxy what actions to take. These actions generally result in instructions to the Fusion engine, responses to the application, or both.

For example, the following stylesheet instruction matches the NewOrder element in the application source document; it then creates a template for an FNCndCommand element in the resulting command document. Using a Command attribute, this element directs the engine to create an instance of FNOrdersProcess:

Find the NewOrder element
Create the FNCndCommand element and its attributes

```
<xsl:template match="NewOrder">
  <FNCndCommand Command="CreateProcess"
                ProcessName="FNOrdersProcess">
    . . .
  </FNCndCommand>
</xsl:template>
```

In the above example, the <FNCndCommand…> tag by itself has the same effect as the <XSL:element> tag shown previously; it creates a new element in the results document.

Outbound Transformations

After a proxy has instructed the process engine to perform an activity, the engine sends a state document to the proxy describing the current state of the activity. An outbound stylesheet transforms this state information into a command document.

Also, you might need to communicate process attribute values to an application. The following template matches an FNProcessAttribute element and transforms it and the information contained in its Name and Type attributes back into Att, AttName, and AttType elements that are part of the application's XML vocabulary. The template also places the value of the FNProcessAttribute into an AttValue element:

Match source element
Create Att element
Create AttName element
Copy value of Name attribute
End tag
Create AttType element
Copy value of Type attribute
End tag
Create AttValue element
Copy FNProcessAttribute value

```
<xsl:template match="FNProcessAttribute">
  <Att>
    <AttName>
      <xsl:value-of select="@Name">
    </AttName>
    <AttType>
      <xsl:value-of select="@Type">
    </AttType>
    <AttValue>
      <xsl:value-of select="text()|*">
    </AttValue>
  </Att>
</xsl:template>
```

In the above example, the "@" indicates an XML attribute. The line "<xsl:value-of select="text()|*">" copies the textual content of the AttValue element to the results document.

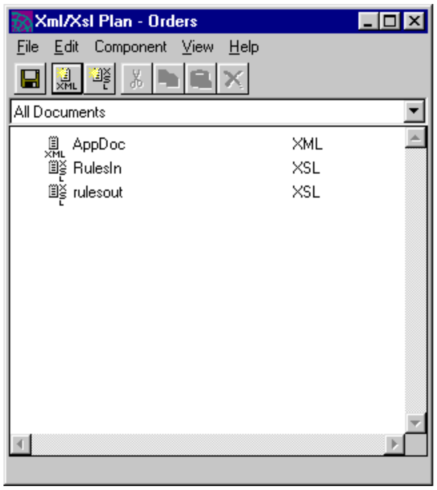
Developing Stylesheets with the Fusion Workshops

Fusion provides two graphical workshops that you can use to create, test, and manage the stylesheets you want to use with a Fusion backbone:

- The XML/XSL Plan Workshop lets you define collections of XML source documents and XSL stylesheets to use for testing and debugging.

  Source documents can be actual application documents or documents that you create for testing purposes. You can create documents directly in the workshop or import existing files. You also can register stylesheets with Fusion backbones directly from this workshop.

  Figure 3 shows an XML/XSL Plan with an XML source document and two XSL stylesheets:

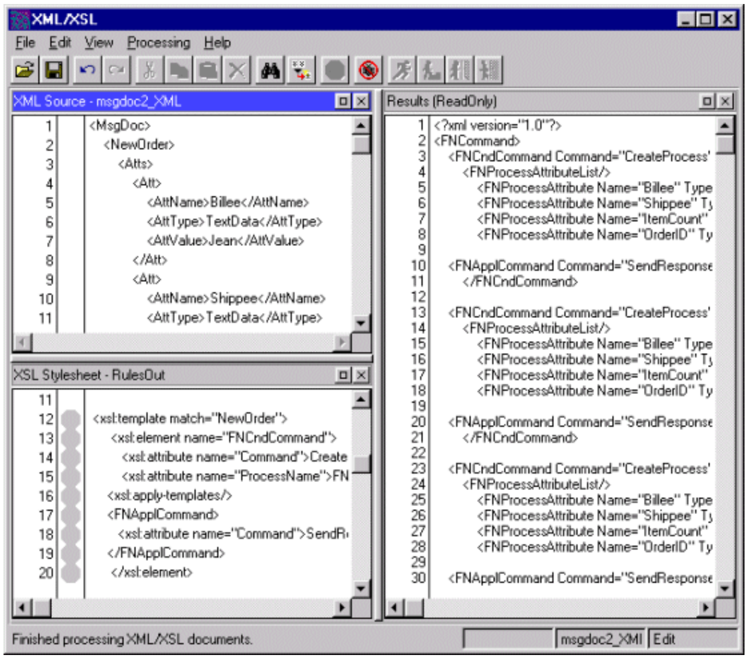
AppDoc	XML
RulesIn	XSL
rulesout	XSL

  Figure 3 *XML/XSL Plan Workshop*

- From the XML/XSL Plan Workshop, you can open the XML/XSL Workshop. This second workshop lets you edit and debug your XML source documents and XSL stylesheets.

  You can validate source documents and stylesheets to ensure that they are well-formed, and process source documents against a stylesheet to generate an XML results document. The workshop debugger makes it easy to find and fix any errors in your stylesheets before you register them. For ease in debugging, you can simultaneously display your XML input, XML output, and the associated XSL stylesheet.

Figure 4 shows the XML/XSL Workshop with an XML source document, XSL stylesheet, and XML results document:



XML Source - msgdoc2_XML

```
<MsgDoc>
   <NewOrder>
      <Atts>
         <Att>
            <AttName>Billee</AttName>
            <AttType>TextData</AttType>
            <AttValue>Jean</AttValue>
         </Att>
         <Att>
            <AttName>Shippee</AttName>
            <AttType>TextData</AttType>
```

XSL Stylesheet - RulesOut

```

<xsl:template match="NewOrder">
    <xsl:element name="FNCndCommand">
       <xsl:attribute name="Command">Create
       <xsl:attribute name="ProcessName">FN
   <xsl:apply-templates/>
   <FNApplCommand>
      <xsl:attribute name="Command">SendR
   </FNApplCommand>
    </xsl:element>
```

Results (ReadOnly)

```
<?xml version="1.0"?>
<FNCommand>
   <FNCndCommand Command="CreateProcess'
      <FNProcessAttributeList/>
         <FNProcessAttribute Name="Billee" Type
         <FNProcessAttribute Name="Shippee" Ty
         <FNProcessAttribute Name="ItemCount"
         <FNProcessAttribute Name="OrderID" Ty

   <FNApplCommand Command="SendResponse
      </FNCndCommand>

   <FNCndCommand Command="CreateProcess'
      <FNProcessAttributeList/>
         <FNProcessAttribute Name="Billee" Type
         <FNProcessAttribute Name="Shippee" Ty
         <FNProcessAttribute Name="ItemCount"
         <FNProcessAttribute Name="OrderID" Ty

   <FNApplCommand Command="SendResponse
      </FNCndCommand>

   <FNCndCommand Command="CreateProcess'
      <FNProcessAttributeList/>
         <FNProcessAttribute Name="Billee" Type
         <FNProcessAttribute Name="Shippee" Ty
         <FNProcessAttribute Name="ItemCount"
         <FNProcessAttribute Name="OrderID" Ty

   <FNApplCommand Command="SendResponse
```

Finished processing XML/XSL documents. | msgdoc2_XMl | Edit

Figure 4 *XML/XSL Workshop*

For information about using the XML/XSL Plan Workshop and the XML/XSL Workshop, see the Fusion Backbone online help.

Developing Fusion Stylesheets: General Guidelines

The following sections provide some general guidelines for writing XSL stylesheets for a Fusion enterprise application. For more information about Fusion XSL support, see the Fusion Backbone online help.

Including Required Declarations and Processing Instructions

The proxy's XSL processor expects the XML declaration and the XSL stylesheet element, containing the XSL namespace declaration, as the first two lines of all stylesheets. The last line of the stylesheet must be the close tag for the stylesheet element.

Include the following declarations in all your Fusion stylesheets:

```
<?xml version="1.0"?>
<xsl:stylesheet http://www.w3.org/1999/XSL/Transform"
             version="1.0">
. . .
</xsl:stylesheet>
```

You can use <xsl:transform> and </xsl:transform> interchangeably with the xsl:stylesheet start and end tags.

When you create your stylesheets in the Fusion XML/XSL Workshop, the required items are added automatically.

Specifying the XML Output Type

When you process an XML source document with an XSL stylesheet, the results document can be any of several types—HTML, plain text, or XML. All Fusion documents must have XML as their type so that the proxy can process them.

To specify that your results documents be XML documents, include the following instruction in each of your stylesheets:

```
<xsl:output method="xml" indent="yes"/>
```

The indent attribute specifies that the lines of the results document be indented to reflect the document's XML hierarchy. Although the proxy does not require indentation to process the document, you can read an indented document more easily in the XML/XSL Workshop or other XML editor.

Creating Elements and Attributes

Many of your applications, as well as Fusion itself, have distinct XML vocabularies. Thus an element or attribute in one vocabulary may have a different name in another vocabulary. Also, one application may store certain values as elements, and another application may store those values as attributes.

Many of the transformations you must write, therefore, involve creating new XML elements and attributes in the results document. For example, Fusion uses an FNProcessAttribute element to indicate a data value. However, an application may use an element called Att to represent a data value. Thus, when you send a message from the proxy to the partner application, you must transform each FNProcessAttribute element into an Att element, so that the client can process the value.

This section describes how to create elements and attributes.

For more information about working with process attributes, see "Handling Process Attributes in Stylesheets" on page 32.

Note An XML attribute is a value attached to an element. A Fusion process attribute is a value defined as part of a process definition. While XML attributes and process attributes can be transformed into each other in a Fusion application, there is no inherent relationship between the two entities.

Elements

Use either of the following methods in your XSL stylesheets to create elements in your results document:

Literal result element

Include the element in the stylesheet as you want it to appear in the results document. The XSL processor copies it to the results document as a literal result element.

To create an element called WorkType in an application document, include the following line in your outbound stylesheet:

Create new element

```
<WorkType/>
```

xsl:element

Use the xsl:element element, specifying the new element name as the value of the Name attribute.

To create the WorkType element using xsl:element, include the following lines in your outbound stylesheet:

Create new element
End new element

```
<xsl:element name="WorkType">
</xsl:element>
```

The two methods are functionally identical, and which one you use is strictly a matter of personal preference. While xsl:element is more verbose, it can make the stylesheet easier to read.

Attributes

Similarly as for elements, there are two methods to create attributes in results documents:

Literal result element

Specify the attribute name and value as literal result elements.

To create the WorkOrder element with a WorkType attribute whose value is “New Work,” include the following lines in your outbound stylesheet:

New element and attribute

```
<WorkOrder WorkType="NewWork"/>
```

xsl:attribute

Use the xsl:attribute element to create the new attribute.

The xsl:attribute element has a required name attribute that specifies the name of the attribute to create. You specify the attribute value as text (with whitespace being significant) within the xsl:attribute element start and end tags.

To create the WorkType attribute, with a value of “NewWork,” as shown in the previous example, include the following lines in your outbound stylesheet:

Create WorkType attribute

```
  <xsl:attribute name="WorkType">NewWork</xsl:attribute>
```

Again, this alternative is more verbose, but it makes the stylesheet more readable. Also, you must use this method if you are creating new attributes for existing elements.

Including Default Templates

As part of its standard implementation, XSLT provides default templates to ensure that all elements of a source XML document are processed by a stylesheet. Even though the Fusion XSL processor observes these defaults, it is good practice to include explicitly the two most important default templates in all your stylesheets. Doing so helps with debugging, and it also ensures that the stylesheets are portable to other XSL processors.

The first of these defaults matches the root element of the source document, then recursively processes all its children, that is, all the nodes of the source document:

Match the root element
Process children of root

```
<xsl:template match="/">
  <xsl:apply-templates/>
<xsl:template>
```

Because the root element is always matched, this template ensures that all child nodes of the root element are processed, even when there are no explicit matches in the stylesheet.

The other default rule you should include processes all text nodes of the source document. This rule is useful to ensure that all data values that are stored in the source document as text are copied to the results document, even if there are not specific matches on the elements that contain these values.

Note There are situations where you do not want all text nodes to be copies to the results document. For information about overriding the default behavior, see the next section, "Overriding the Default for Text Nodes."

The default rule for text nodes concatenates the values of all text nodes and copies them to the results document:

Match any text node
Copy value of current node

```
<xsl:template match="text()|@*">
  <xsl:value-of "."/>
</xsl:template>
```

Note The "@*" in the above template, indicating a match for any attribute node, is shown for compliance with the XSLT Recommendation. Because attributes technically are not child nodes, however, they are never implicitly matched by the first default template shown above. Therefore, you can omit the match for attributes, and include the match for text nodes only.

Overriding the Default for Text Nodes

You often do not want all text nodes copied to the results document. Rather, you only want the text that your stylesheet explicitly matches.

For example, if your application document contains multiple customer orders, the default rule for text nodes might cause the data from all orders to be copied to the results document. However, you might only want to process only selected orders and ignore the others.

To copy text nodes selectively to the results document:

1 Include the following template in place of the usual default for text nodes:

```
<xsl:template match="text()"/>
```

This “empty” template matches any text node. However, rather than performing some action on the match, as the default template does, it simply ends (with the “/”), doing nothing.

Thus, rather than copying the text to the results document, as the default rule does, the template simply ignores the match.

2 Include templates in your stylesheet to match the specific text you want to appear in the results document.

Combining Stylesheets

If some of your stylesheets repeat the same transformations, you can create one or more stylesheets with common transformations, then use these stylesheets as part of the stylesheets you write for specific proxies.

XSLT provides two elements for using a stylesheet within another stylesheet:

- xsl:import
- xsl:include

The following subsections describe how to use these elements.

Importing Subordinate Stylesheets

To import a stylesheet as a subordinate of your main stylesheet, use the following element:

```
<xsl:import href="imported_stylesheet"/>
```

where *imported_stylesheet* is an absolute pathname or a pathname relative to the current stylesheet.

You must place the xsl:import tag at the top level of the stylesheet (as a child of the root element), immediately after the xsl:stylesheet declaration.

When there are conflicting templates between the main stylesheet and the imported stylesheet, the templates in the main stylesheet always take precedence. If you import multiple stylesheets, each new import overrides the earlier imports, with the importing stylesheet having the highest precedence of all.

Including External Stylesheets

To include external stylesheets without concern for precedence, use the following element:

```
<xsl:include href="included_stylesheet"/>
```

where *included_stylesheet* is an absolute pathname or a pathname relative to your main stylesheet.

The templates in an included stylesheet are evaluated as though they were part of the main stylesheet. Where there are conflicting templates, the normal rules of priority apply. For information about template conflict resolution, see the XSLT Recommendation at *http://www.w3.org/TR/xslt#conflict*.

The Fusion processor resolves conflicts by using the template that occurs last in the stylesheet.

The xsl:include element must be at the top level of the main stylesheet (as a child of the root element). While there are no restrictions on where in the stylesheet you place the xsl:include element, it is good practice to put it right after the xsl:stylesheet element for readability.

Reusing Templates

In addition to reusing entire stylesheets, you can name individual templates and reuse them later in a stylesheet. For example, you might create a template that sends a specific message to an application, then invoke this template when different elements in the application document are matched.

To name a template, include a name value in the “name” attribute of the xsl:template element. The following example creates a template named “AppMsg” that sends a message to an application:

Name the template
Send application a command
Create the message

```
<xsl:template name="AppMsg"/>
  <FNAplCommand Command="SendMessage" Method="Post">
    <FNMessage>
      lengthy message goes here...
    </FNMessage>
  </FNAplCommand>
</xsl:template>
```

To invoke this template, use the xsl:call-template element within any desired xsl:template element:

Match an element
Call the named template

```
<xsl:template match="MsgDoc">
  <xsl:call-template name="AppMsg"/>
</xsl:template>
```

Performing Common Fusion Stylesheet Transformations

A primary task involved in integrating your applications with Fusion is to write the XSL stylesheets that transform your proxy documents. For example, you must transform an application document containing a new customer order into a command document to start a Fusion process. For each use case in this manual, detailed procedures are provided for writing the specific XSL transformations required for the successful completion of the scenario.

There are a number of transformations, however, that are common to many of the stylesheets you write as part of a Fusion enterprise application. The following sections provide instructions for including some of these transformations in your stylesheets.

Handling Process Attributes in Stylesheets

When you create a Fusion process definition, you define process attributes to hold values that are used by the applications in the process. In a Fusion enterprise application, both inbound and outbound stylesheets often need to manipulate Fusion process attributes:

- Inbound stylesheets take values from an application document (or supply values themselves) and place them into Fusion process attributes.
- Outbound stylesheets retrieve process attribute values from state documents and can send them to an application.

The following sections describe how your stylesheets can generate and retrieve process attributes. For more information about creating process attributes in Fusion, see the *Forte Fusion Process Development Guide*.

Generating Process Attributes

Inbound stylesheets often need to generate Fusion process attributes so that processes can be started and activities completed. To declare process attributes, include the following elements in an inbound stylesheet:

- FNProcessAttributeList to define a list of process attributes
- FNProcessAttribute to define a process attribute

  The FNProcessAttribute element has required Name and Type attributes to specify the process attribute name and data type.

For Fusion to use any process attributes you generate through a stylesheet, the process attribute must have been created as part of the Fusion process definition, and the names and data types must match those you specify in the Name and Type attributes of the FNProcessAttribute element. Fusion ignores any process attributes created by a stylesheet if the process attributes are not known to the process engine.

For an application to return an updated value for a process attribute, the process attribute must have been defined with a lock type of Write or WriteQueue in the Fusion process definition. For information about lock types, see the *Forte Fusion Process Development Guide*.

Process Attribute Creation Example

The following example matches the Customer element in an application document, then creates a list with two process attributes based on values found in the application document:

- The CustName process attribute derives its value from the CustomerName child of the Customer element.
- The CustID process attribute derives its value from the CustomerID attribute of the Customer element.

Match Customer element
Create process attribute list
Create process attribute
Get value from source element.

Create process attribute
Get value from source attribute

```
<xsl:template match="Customer">
  <FNProcessAttributeList>
    <FNProcessAttribute Name="CustName" Type="TextData">
      <xsl:value-of select="CustomerName">
    </FNProcessAttribute>
    <FNProcessAttribute Name="CustID" Type="TextData">
      <xsl:value-of select="@CustomerID">
    </FNProcessAttribute>
  </FNProcessAttributeList>
</xsl:template>
```

A stylesheet not only can create a process attribute, but also can set its value, rather than obtaining the value from the application. The following example creates a process attribute called OrderStatus, and supplies a value of “New Order”:

Create the process attribute
Define process attribute’s value

```
<FNProcessAttribute Name="OrderStatus" Type="TextData">
  New Order
</FNProcessAttribute>
```

Transforming Process Attribute Lists

The state document for an activity also contains the current values for any process attributes specified in the application dictionary entry for the activity. The process attribute list has the following structure:

```
<FNProcessAttributeList>
  <FNProcessAttribute Name="proc_att_name" Type="data_type">
    attribute_value
  </FNProcessAttribute>
  additional process attributes...
</FNProcessAttributeList>
```

You often need to provide these values to the application, for example, when requesting a service provider to start an activity. To do so, your outbound stylesheet must transform the process attribute list in the state document into a vocabulary and structure that the application understands.

In the example application, the process attribute list is transformed into an element called Atts in the application document. Each process attribute corresponds to a child element called Att. Finally, the name, data type, and value of each process attribute correspond to the AttName, AttType, and AttValue child elements of the Att element.

Your outbound stylesheet must provide two templates to perform these transformations:

- a template to transform the FNProcessAttributeList element into the Atts element
- a template to transform each FNProcessAttribute element and its attributes into an Att element and its child elements

Process attribute list

To transform a process attribute list for an application document, include a template like the following in an outbound stylesheet:

Match process attribute list
Create app doc list element
Process children

```
<xsl:template match="FNProcessAttributeList">
  <Atts>
    <xsl:apply-templates/>
  </Atts>
</xsl:template>
```

Process attributes

To transform a process attribute for an application document, include a template like the following in an outbound stylesheet:

Match process attribute element
Create Att element in app doc
Create AttName child element
Copy value of Name attribute
Create AttType child element
Copy value of Type attribute
Create AttValue child element
Copy value of process attribute

```
<xsl:template match="FNProcessAttribute">

  <Att>
    <AttName>
      <xsl:value-of select="@Name">
    </AttName>
    <AttType>
      <xsl:value-of select="@Type">
    </AttType>
    <AttValue>
      <xsl:value-of select="text()|*">
    </AttValue>
  </Att>
</xsl:template>
```

Transformation Notes

In the above template:

- The values of the newly generated AttName and AttType elements are supplied by copying the values of the FNProcessAttribute element's Name and Type attributes.
- To place the process attribute value into the AttValue element, the template copies any text nodes found within the FNProcessAttribute element.

Transforming Process Attribute Values

Application document elements or attributes and their corresponding Fusion process attributes might have different data formats as well as different names. Also, applications within the same process might represent names differently, or use different units of measurement or currency.

For example, customer John Smith might be known to one application as follows:

```
<CustomerName>
  <FirstName>John </FirstName>
  <LastName>Smith</LastName>
</CustomerName>
```

However, the corresponding Fusion process attribute might represent customer John Smith as follows:

<CustName>John Smith</CustName>

Thus, when your inbound stylesheet generates the process attribute, it also must change the way the value is represented. When the outbound stylesheet sends the process attribute value back to the application, it also must change its representation back to what the application understands.

XSLT, through XPath, provides a wide variety of functions for manipulating character and numerical data. In the following example shown here, when you generate the CustName element, you also must use a concatenation function to combine the first and last names of the customer.

To accomplish this task, use a template like the following:

Create the process attribute
Combine app. element values

```
<FNProcessAttribute Name="CustName" Type="TextData">
  <xsl:value-of select="concat(FirstName,LastName)">
</FNProcessAttribute>
```

To transform the customer name in the opposite direction, apply another XPath string function that deconstructs the single name value into first and last names.

For information on the full set of functions available, see the XPath recommendation at *http://www.w3c.org*.

Transmitting Application Documents as Process Attributes

You might want the application document that initiates a Fusion process to be transmitted as a Fusion process attribute to additional applications in the process. In this way, each application can share a business object, such as a customer order or employee record.

To transmit an application document as a process attribute:

1. In your Fusion process definition, declare a process attribute of type XmlData to hold the application document.

   For greatest run-time efficiency, create a read-only process attribute to hold the application document. That is, do not modify the original document as it moves through the process (in much the same way that you would pass an original paper document such as a purchase order unchanged through a manual process).

2. In your inbound stylesheet that processes the application document, generate the process attribute defined in Step 1.

3. Use an xsl:copy-of element to match the root element of the application document, then copy all its children (that is, the entire document) into the process attribute you created in Step 2.

The following stylesheet transformations perform Step 2 and Step 3:

Create the process attribute
Select root; copy to result tree

```
<FNProcessAttribute Name="CustOrder" Type="XMLData">
  <xsl:copy-of select="/">
</FNProcessAttribute>
```

If you want subsequent applications in the process to modify parts of the original document, for example, order status or completion date, create individual Fusion process attributes with Write or Write Queue locks. To pass the information as an XML document, create the process attribute with the XmlData data type.

Supplying Activity Information to Applications

The Fusion process engine generates a state document with information about an activity who state has changed to ACTIVE. This information is contained in an FNIdentity element, each attribute of which specifies information about the activity, for example, the process ID or activity name.

For a complete description of the FNIdentity element and its attributes, see the Fusion Backbone online help. For information about activity states, see the *Forte Fusion Process Development Guide*.

Your outbound stylesheets can retrieve the value of an FNIdentity attribute from the state document, then present it in a format that is meaningful to the application.

▶ **To supply activity information to an application:**

1. Include a template that matches the value of the desired attribute of the FNIdentity element.
2. Create a new element that the application understands.
3. Copy the value of the attribute into the new element.

The following example matches the ProcessID attribute of the FNIdentity element and passes its value to the partner application as the order confirmation number:

Match relevant State attribute of the FNIdentity element

Create the Cfnumber element

Give it value of process ID

```
<xsl:template
match="FNCndState[@State=’ProcessStarted’]/FNIdentity">
  <Cfnumber>
    <xsl:value-of select="@ProcessID"/>
  </Cfnumber>
</xsl:template>
```

Specifically, the template in this example uses an XPath location path to match the desired value in the state document. This location path specifies an FNIdentity element that is a child of a FNCndState element with a State attribute (indicated by “@State”) whose value is “ProcessStarted.” The xsl:value-of element copies the value of the ProcessID attribute into the new Cfnumber element.

For more information about the structure and vocabulary of state documents, see the Fusion Backbone online help.

After you create the elements you want, you then can pass them to an application as described in the following section, “Sending Messages to Applications.”

Sending Messages to Applications

Both inbound and outbound stylesheets can specify that messages be sent to applications. An inbound stylesheet might generate a message to let the application know that the application document has been received. An outbound stylesheet might generate a message based on the state of an activity, for example, that the activity has been started.

Your stylesheets thus need to include rules for generating such messages. You use the following elements in your stylesheets to generate messages to applications:

- FNAplCommand to specify how to send the message
- FNMessage to indicate the start and end of the message

  Whatever you include between the start and end tags of the FNMessage element is sent as the body of the message.

Specifying the Message Type

Use the Command attribute of the FNAplCommand element to specify how to send the message to the application. This attribute can take either of these values:

- SendResponse to send an HTTP message in response to the application's message
- SendMessage to send a new HTTP message and wait for a response

When you use SendMessage, specify either of the following HTTP methods for sending the message content:

- Get appends the contents of the message to the URL used to contact the application.
- Post includes the contents of the message in the body of the HTTP message, so that it is not visible as part of the URL.

Post is the more secure method. However, the application must be capable of receiving messages sent with the Post method.

For the complete syntax of the FNAplCommand element, see the Fusion Backbone online help.

JMS Considerations

Proxies configured for JMS also use the FNAplCommand to generate messages to applications, but be aware of the following special considerations:

- If you use SendResponse to respond to an incoming JMS message, then the body of the XML must be empty.
- The method (Get or Post) must be specified, even though it is ignored by proxies configured for JMS.

Completing a Roundtrip Communication

The HTTP communication protocol that Fusion uses is a *request/response* protocol. That is, when an HTTP client sends a request to an HTTP server, the server must respond to the client in some way, even if only to acknowledge receipt of the request.

For example, when you enter a URL into a Web browser (an HTTP client), the Web server (HTTP server) responds by displaying the page. If the server cannot find the page, it must send a message to this effect to the browser.

Note JMS is inherently an asynchronous communication protocol. This section on round-trip communication applies only to proxies configured for HTTP.

HTTP messages can have two parts:

- a header containing metadata about the message, such as the recipient's name and location
- a message body with the contents of the message, such as the customer order

Every HTTP message has a header; the body is optional. You often use an HTTP header with no body, known as an *empty message*, when you have no information to transmit, but you simply need to complete an HTTP communication cycle.

In Fusion, when a partner application sends an application document to the proxy, the proxy must complete the communication in either of these ways:

- The outbound stylesheet generates an FNAplCommand element whose Command attribute has a value of "SendMessage." The FNAplCommand element contains an FNMessage element that contains an application document to send to the application.

- The inbound stylesheet generates a response in the form of an empty FNAplCommand element whose Command attribute has a value of “SendResponse.” In this case, the proxy sends a message header only, with no message content.

  When the proxy generates such a response, any subsequent message from the application, for example a message that the activity has been performed, constitutes the beginning of a new roundtrip communication. To complete this communication, your inbound stylesheet must have another template to generate a response to this second application document. (This response also could be in the form of an empty FNAplCommand element.)

  For an example of this message-response cycle, see Chapter 4, “Asynchronous Service Provider Use Case.”

For more information about the HTTP protocol, see the HTTP specification at http://www.w3.org/Protocols/.

Message Examples

The following examples show templates you can include in stylesheets to send messages to applications. The examples assume you have a proxy configured for HTTP communication. The differences for proxies configured for JMS communication are called out in notes.

The first example, from an inbound stylesheet, matches the NewOrder element in the application document, then sends a simple SendResponse message to let the application know the order has been received. Because the message has no content, this purpose is implied and would need to be embedded in the application itself.

Note If you use SendResponse to respond to an incoming JMS message, then the XML body must be blank. JMS is an asynchronous protocol, and does not expect a response.

Match NewOrder element
Send HTTP response

```
<xsl:template match="NewOrder">
  <FNAplCommand Command="SendResponse"/>
</xsl:template>
```

The second example, from an outbound stylesheet:

- matches an activity, indicated by the FNCndState element, whose state is specified as “ProcessStarted” in the state document sent by the Fusion process engine
- generates a command to send an application document to the partner application
- generates the application document

Note If your proxy is configured to use JMS, you must still specify the Method attribute, even though the attribute is ignored.

Match the activity state
Send an HTTP message
Start the application document
Document element of app doc
Message content

```
<xsl:template match="FNCndState[@State=’ProcessStarted’]">
  <FNAplCommand Command="SendMessage" Method="Post">
    <FNMessage>
      <OrderConfirmation>
        OrderEntered
      </OrderConfirmation>
    </FNMessage>
  </FNAplCommand>
</xsl:template>
```

Communicating with Non-Partner Applications

When a stylesheet causes a message to be sent to an application, the message is sent to the proxy's partner application by default. However, you might want the proxy to send a message to another application as well. For example, you might want a shipping application to be notified when each new order has been placed.

To send a message to an application other than a proxy's partner, use the following elements, which are children of the FNAplCommand element:

- FNDestination in your stylesheet to specify the address of the alternate application
- FNMessage to embed the message

For example:

Command to send message
Application address
Start the message
document element of app doc
Message contents

```
<FNAplCommand Command="SendMessage" Method="Post">
  <FNDestination Address="canus.dogstar.com:120"/>
  <FNMessage>
      <OrderConfirmation>
        order number 123 placed on 3/3/00
      </OrderConfirmation>
  </FNMessage>
</FNAplCommand>
```

Communicating Between Applications Without A Process Definition

In some relatively simple integration scenarios, you might want applications to communicate with each other through a Fusion backbone, but not use the process management capabilities that the Fusion process engine provides. For example, you might want a personnel application to provide new employee information to other applications, such as a defect tracking system. You can use Fusion for this purpose without having to define a Fusion process or run a process engine.

A proxy that functions without interacting with a Fusion process engine is called an *independent proxy.* An independent proxy:

- operates as a server (or client/server) because the partner application is responsible for initiating work
- receives an application document from its partner application document
- uses an inbound stylesheet to generate a command document that sends an application document to one or more target applications

  Because an independent proxy never receives a state document from a Fusion process engine, it does not require an outbound stylesheet. Rather, the inbound stylesheet generates the commands to communicate with the target application, as described in the next section. For an example of a stylesheet that an independent proxy could use, see *nopein.xsl* in the TOOL adapter example program directory.

Creating an Inbound Stylesheet

The inbound stylesheet for an independent proxy must match a pattern in the source document, for example, the document element that defines an employee record. The stylesheet then must generate the following elements in the command document:

- FNCommand to indicate a command document
- FNAplCommand to specify that a command should be sent to an application
- FNDestination to specify the URL of any target application other than the default partner configured for the proxy

  If there is more than one target application, the stylesheet must provide an appropriate match that generates an FNDestination element for each target application.
- FNMessage to encapsulate the application document

  Any information between the FNMessage start and end tags becomes the application document that is sent to the target application.

  With multiple target applications, you typically need multiple FNMessage elements (one to correspond to each FNAplCommand element) to create the appropriate application document tailored to each application's information needs and XML structure and vocabulary. An FNAplCommand message is also needed for responding to the original request.

For more information about using these elements, see "Performing Common Fusion Stylesheet Transformations" on page 32 and the Fusion Backbone online help.

For use cases involving independent proxies, see Chapter 9, "Independent Proxy Use Case" and Chapter 10, "Independent Proxy Authentication Use Case."

Configuring an Independent Proxy

When you configure an independent proxy, use the **UseProcessEngine** command and set process engine usage to **off**.

If the proxy requires HTTP user authentication from the requestor application, use the following FNScript commands:

- For the **SetAuthentication** command, set the **Scheme** parameter to **Basic** and the **Server** parameter to **Local**.
- Use the **SetCredentials** command to specify the user name and password the proxy expects from the application.
- Use **SetPort** to configure the port.

For more information about configuring proxies and specifying user authentication, see the *Forte Fusion Backbone System Guide*.

Omitting Fusion Process Management Functions

Use independent proxies for situations that have simple application integration requirements. Without a connection to a Fusion process engine, a Fusion application cannot make use of various process management features:

- Because there is no process definition, you cannot specify the timing of tasks.
- You cannot use Fusion process attributes to store values that are shared by different applications.

  Your stylesheet is responsible for each value that you send between applications. You also cannot store and transmit application documents as XmlData process attributes.
- You have limited HTTP user authentication functions:
  - You must use Basic authentication, which lets the independent proxy require a user name and password when the requestor application initiates a session. Using a Fusion process definition allows FusionXML authentication, for specifying additional user information.
  - You cannot make use of such Fusion user identification features as User Validations, User Profiles, and Roles.
- Without a Fusion process engine, the proxy has no way to maintain state information about the current status of an activity. Thus, if the proxy or application fails, there is no way to recover information about which tasks are current or completed.

If your Fusion enterprise application is complex enough to require some number of these features, you should redesign it to use a Fusion process engine.

Chapter 2

Service Requestor Use Case

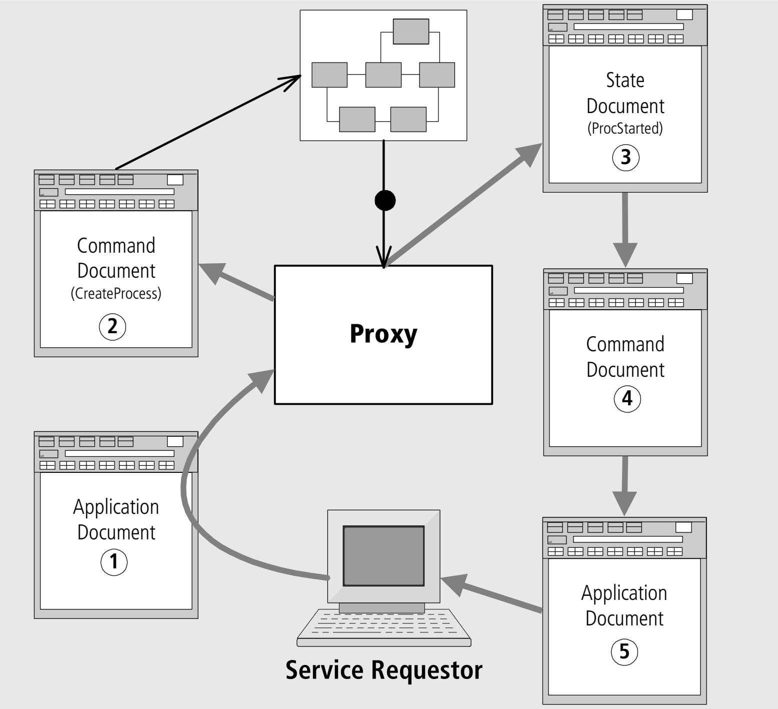
This chapter describes the XSL stylesheets you need to write and related integration tasks you need to perform to enable a service requestor application to initiate a Fusion process.

If the proxy requires authentication from the service requestor before creating the process, there are additional tasks you must perform. After you read this chapter, see Chapter 5, "Service Requestor Authentication Use Case."

For more information about the way in which Fusion processes requests from service requestors, see Appendix A, "Transforming Proxy Documents."

Use Case Summary

The following table provides an overview of this use case:

Use Case Information	Description
description	A service requestor application attempts to initiate a Fusion process.
expected outcome	The application's proxy instructs the process engine to start the process, and the proxy notifies the application that the process has been started.
actors	▮ service requestor application ▮ application proxy ▮ Fusion process engine
proxy document flow	**1** Service requestor sends application document to proxy. **2** Proxy sends command document to engine to start the process. **3** Proxy creates state document based on information from engine about process. **4** Proxy generates command document that includes message for application that request was received. **5** Proxy sends application document to service requestor.  State Document (ProcStarted) 3 Command Document (CreateProcess) 2 Proxy Command Document 4 Application Document 1 Application Document 5 Service Requestor
required stylesheets	inbound stylesheet to transform application document into command document to start process (Step 2) outbound stylesheet to: ▮ transform state document into command document to notify application (Step 4) ▮ transform message embedded in command document into application document containing notification (Step 5)
other integration tasks	▮ ensure that any Fusion process attributes referred to in the inbound command document are part of the process definition ▮ use the **AddListener** FNscript command to configure a proxy as a listener (HTTP server or JMS listener) ▮ you can also use the **SetPort** FNscript command to configure the proxy as an HTTP server

The remainder of this chapter describes this use case in more detail and provides procedures for writing the stylesheets required for its successful completion.

Starting the Fusion Process

Each Fusion process has one application, called the *service requestor*, that initiates the process. In the example application, the order entry application is the service requestor.

When a service requestor sends an application document to its proxy, the proxy must generate a command document to the Fusion process engine. The command document:

- contains the FNCommand document element to identify itself as a command document
- instructs the engine to start the process
- passes any required process attributes (and possibly other process attributes) to the engine

The proxy's inbound stylesheet must contain the rules that cause this command document to be generated correctly. The following subsections describe how to include these rules in your inbound stylesheet.

Authentication

The following steps assume that any user is authorized to start a process. For information about how to allow only authorized users to create a process, see Chapter 5, "Service Requestor Authentication Use Case."

Configuration Notes

Connecting to JMS provider application

For proxies configured to use JMS, use the **SetProtocol Name=JMS** Fusion Script command to specify information for connecting to a JMS provider application.

Adding listeners

To enable the proxy to receive requests from the partner application, configure the client as a listener (HTTP server or JMS listener) using the **AddListener** Fusion Script command. When issuing this command specify the protocol you are using for messaging (HTTP or JMS) and other information specific to the protocol.

If you specify HTTP, then you specify the port at which the proxy should receive messages from an application.

If specify JMS, then you specify the messaging model (point-to-point or publish/subscribe) and other optional information for JMS messaging (JMS message selectors, acknowledgement mode, and durable messaging behavior).

You can also use the **SetPort** command to configure an HTTP listener.

For more information about configuring proxies, including documentation on Fusion Script commands, see the *Forte Fusion Backbone System Guide*.

Step 1. Generating the Document Element

An application proxy identifies a command document using the FNCommand element as its document element. The first task your inbound stylesheet must perform is to generate this document element.

To generate the FNCommand document element, include the following template in your inbound stylesheet:

Match app doc root element

Create command doc doc elem.

Process children of root element

```
<xsl:template match="/">
  <FNCommand>
    <xsl:apply-templates/>
  </FNCommand>
</xsl:template>
```

Transformation Notes

In the above template:

- The match could also have been on the document element of the application document. It is good practice, however, to match the root element, because this match is guaranteed to be successful.
- The xsl:apply-templates element ensures that the XSLT processor processes all children of the root element of the application document. Without the xsl:apply-templates element, the XSLT processor would complete its work, and the FNCommand element would be generated as an empty element.

Step 2. Instructing the Engine to Start the Process

Your inbound stylesheet next must generate a command to the Fusion process engine to start the process. This command is contained in the FNCndCommand element, which has two attributes:

- Command tells the engine what action to take, in this case, to start a process.
- ProcessName tells the engine the name of the process to start.

To generate the appropriate FNCndCommand element and attributes, include a template similar to the following in your inbound stylesheet:

Match NewOrder elem. in app doc

Send CreateProcess command and process name

Process children of NewOrder

```
<xsl:template match="NewOrder">
  <FNCndCommand Command="CreateProcess"
              ProcessName="FNOrdersProcess">
    <xsl:apply-templates/>
  </FNCndCommand>
</xsl:template>
```

Transformation Notes

In the above template:

- The xsl:apply-templates element continues the recursive processing begun in Step 1 under Starting the Fusion Process on page 46. Thus, the FNCndCommand element is generated within its parent FNCommand element, and the elements you create in the next step become children of FNCndCommand.
- The stylesheet supplies both the command for the engine and the name of the process to start. Thus, the application does not need to have knowledge of this information.

Step 3. Passing Process Attributes to the Engine

The next step in starting the process is to supply the Fusion process engine with values for the required process attributes. This task is accomplished by:

- creating a process attribute list, indicated by an FNProcessAttributeList element
- creating each process attribute, indicated by an FNProcessAttribute element
- copying the appropriate values from the application document

For procedures for creating process attributes, see “Generating Process Attributes” on page 32.

Your process definition can contain process attributes of type XmlData that contain well-formed XML documents as their values. In the example application, the entire application document is passed as the value of the StartingMessage process attribute. You also could pass part of the application document as a process attribute in addition t, or instead of, the entire application document.

For information on including an application document in a process attribute, see “Transmitting Application Documents as Process Attributes” on page 35.

Command Document Example

The following command document, generated from the service requestor’s application document, is created by the preceding steps.

```
Doc element for cmd        <FNCommand>
doc
Command to engine to         <FNCndCommand Command="CreateProcess"
  create process                             ProcessName="FNOrdersProcess">
Start proc attribute list      <FNProcessAttributeList>
Process attribute                <FNProcessAttribute Name="Billee"
                                                   Type="TextData">Jack</FNProcessAttribute>
Process attribute                <FNProcessAttribute Name="Shippee"
                                                   Type="TextData">Jill</FNProcessAttribute>
Process attribute                <FNProcessAttribute Name="ItemCount"
                                                   Type="TextData">12</FNProcessAttribute>
Process attribute                <FNProcessAttribute Name="OrderID"
                                                   Type="TextData">555</FNProcessAttribute>
App doc as process               <FNProcessAttribute Name="StartingMessage" Type="XmlData">
  attribute
App doc doc element                <MsgDoc>
                                     <NewOrder>
                                       <Atts>
                                         <Att>
                                            <AttName>Billee</AttName>
                                            <AttType>TextData</AttType>
                                            <AttValue>Jack</AttValue>
                                         </Att>
                                         <Att>
                                            <AttName>Shippee</AttName>
                                            <AttType>TextData</AttType>
                                            <AttValue>Jill</AttValue>
                                         </Att>
                                         <Att>
                                            <AttName>ItemCount</AttName>
                                            <AttType>TextData</AttType>
                                            <AttValue>12</AttValue>
                                         </Att>
                                         <Att>
                                            <AttName>OrderID</AttName>
                                            <AttType>TextData</AttType>
                                            <AttValue>555</AttValue>
                                         </Att>
                                       <Atts>
                                     </NewOrder>
End of application doc             </MsgDoc>
                                 </FNProcessAttribute>
End process attrib list        </FNProcessAttributeList>
                             </FNCndCommand>
                           </FNCommand>
```

Document Notes

In the above command document:

- The application document is sent to the engine as a process attribute of type XmlData. This step is somewhat redundant in this case and is shown for example purposes only. The information about the order that the engine actually uses is contained in the other process attributes, which contain the same information as the application document.
- The CDATA sections shown in the actual XML that is generated when you run the example application are omitted here for the sake of simplicity. CDATA sections allow you to include characters that are generally processed as XML markup, such as "<" or ">", without having the processor replace them with entities, such as "&lt;" or "&gt;".

Transforming State Information

When the process engine notifies the proxy that the process was started, the proxy generates a state document that contains:

- the state (“ProcessStarted”) of the new process
- the process name and ID
- the current value of the process attributes used by the process

The proxy’s outbound stylesheet must include templates that cause the following actions to take place:

- The proxy generates a command document with a command to send the application a message.
- The proxy generates the application document containing the appropriate message.

The following subsections describes how to include these rules in your outbound stylesheet.

Step 1. Generating the Document Element

The first step the outbound stylesheet must perform is to generate the FNCommand element that designates a command document. The template for this task in the outbound stylesheet is the same as in the inbound stylesheet.

For instructions for generating the document element for a command document, see “Generating the Document Element” on page 46.

Step 2. Generating the Application Command

The next step your outbound stylesheet must perform is to generate the commands to send a message to the application.

▶ **To generate these commands:**

1. Create a template that matches a FNCndState element in the state document whose State attribute has a value of “ProcessStarted.”
2. Generate an FNAplCommand element whose Command attribute has a value of “SendMessage” and whose Method attribute specifies the HTTP method (Get or Post).

Note If your proxy is configured to use JMS, you must still specify an HTTP method, even though this method is ignored when using JMS.

3. Generate an FNMessage element to hold the message to the application.

To generate the appropriate FNAplCommand element and attributes, include a template like the following in your outbound stylesheet

Match an activity whose state is “ProcessStarted”

Send an HTTP message

Start the message

See Step 3 for these rules

```
<xsl:template
match="/FNState/FNCndState[@State=’ProcessStarted’]">
  <FNAplCommand Command="SendMessage" Method="Post">
    <FNMessage>
      rules to generate message content
    </FNMessage>
  </FNAplCommand>
</xsl:template>
```

Transformation Notes

In the above template:

- The full /FNState/FNCndState[@State='ProcessStarted'] location path is used for example purposes only. The parent (FNState) element is not necessary, because the source document contains no other FNCndState elements with different parents (that is, the FNCndState element always appears as a child of the FNState element in a state document).

  For more information about specifying location paths, see the Fusion Backbone online Help and the XPath Recommendation at *http://www.w3.org*.

- The xsl:apply-templates element ensures that the message content (to be generated by the template in the next section) is placed within the FNMessage element.

  For more information about sending messages to applications, see "Sending Messages to Applications" on page 36.

Step 3. Creating the Message Content

After you create the FNMessage element, you create its contents. Whatever is contained between the start and end tags of the FNMessage element is returned as an application document to the service requestor.

The outbound application document can contain whatever information the application expects. In the example application, the application document contains:

- an OrderEntered element, which indicates in the application's XML vocabulary that the order was placed successfully
- a confirmation number, which is generated from the value of the Process ID

  For a template that generates the confirmation number, see the next section, "Generating Values for the Application."

To generate the OrderEntered element, include the following transformations within the template from the previous step:

Message created in Step 2
Create element for app doc
Process children

```
<FNMessage>
  <OrderEntered>
      <xsl:apply-templates/>
  </OrderEntered>
</FNMessage>
```

Transformation Notes

In the above template:

- The transformation is a continuation of the one shown in the previous section, "Generating the Application Command." The templates are broken out here for illustration purposes, but your stylesheet would contain only one FNMessage element to generate this application document.
- The xsl:apply-templates element ensures that the confirmation number element (shown in the next section) is created as a child of the OrderEntered element.

Step 4. Generating Values for the Application

The final step in creating the application document is to pass back to the application any information it needs. You do so by transforming values in the state document, which might have no meaning to the application in their present form, into a different form that the application can understand.

In the example application, the Process ID is transformed into a confirmation number required by the order entry application. Whereas the Process ID itself is meaningless to the application, it provides a unique value that can be transformed easily into the value of an element, the confirmation number, that is meaningful to the application.

To generate the confirmation number, your outbound stylesheet must:

- create the Cfnumber element
- copy the Process ID as the value of the Cfnumber element

To accomplish these tasks, include a template like the following in your outbound stylesheet:

Match FNIdentity child of relevant FNCndState element

Create element for app doc

Copy value of process ID attribute

```
<xsl:template
match="FNCndState[@State=’ProcessStarted’]/FNIdentity">
  <Cfnumber>
    <xsl:value-of select="@ProcessID">
  </Cfnumber>
</xsl:template>
```

Transformation Notes

In the above template:

- The template matches the FNIdentity child element of an FNCndState element whose State attribute has a value of “ProcessStarted.” The FNIdentity element conveys identifying process information in the state document.
- The xsl:value-of element selects the value of the ProcessID attribute of the FNIdentity element and copies this value to the results document as the text-node child of the Cfnumber element.

For more information about extracting values from FNIdentity attributes, see “Supplying Activity Information to Applications” on page 36.

Generated Document Examples

The following sections provide examples of the XML documents that a proxy generates when a Fusion process starts.

State Document

A proxy generates a state document similar to the following after the process engine informs it that a process has been started:

XML declaration

State doc document element
State of process
Process information

App dict process attribute list
StartingMessage proc attribute
Orig service requestor app doc

```
<?xml version="1.0"?>

<FNState>
  <FNCndState State="ProcessStarted">
    <FNIdentity
        ProcessID="52"
          ProcessName="FNOrdersProcess"/>
    <FNProcessAttributeList>
      <FNProcessAttribute Name="StartingMessage" Type="TextData">
        <MsgDoc>
          <NewOrder>
          <Atts>
          <Att>
            <AttName>Billee</AttName>
            <AttType>TextData</AttType>
            <AttValue><![CDATA[fred]]></AttValue>
             </Att>
          <Att>
            <AttName>Shippee</AttName>
            <AttType>TextData</AttType>
              <AttValue><![CDATA[fred]]></AttValue>
          </Att>
          <Att>
            <AttName>ItemCount</AttName>
            <AttType>TextData</AttType>
            <AttValue><![CDATA[20]]></AttValue>
          </Att>
             <Att>
            <AttName>OrderID</AttName>
            <AttType>TextData</AttType>
            <AttValue><![CDATA[15]]></AttValue>
             </Att>
          </Atts>
          </NewOrder>
        </MsgDoc>
      </FNProcessAttribute>
    </FNProcessAttributeList>
  </FNCndState>
</FNState>
```

Command Document

The following command document is generated from the above state document by the transformations described in this chapter.

Start command document
Command to send msg to app
Application document follows
Document element of app doc
Confirmation number element
Process ID as confirmation nbr

Application doc completed

```
<FNCommand>
  <FNAplCommand Command="SendMessage" Method="Post">
    <FNMessage>
      <OrderEntered>
        <Cfnumber>
          1234
        </Cfnumber>
      </OrderEntered>
    </FNMessage>
  </FNAplCommand>
</FNCommand>
```

Application Document

The lines between the start and end tags of the FNMessage element in the above command document make up the application document that is sent to the service requestor as the response to its initial message.

Chapter 3

Synchronous Service Provider Use Case

This chapter describes the XSL stylesheets and related integration tasks that enable a proxy to initiate a synchronous session with a service provider application. In this case, the proxy initiates an HTTP session by sending a request for work to a service provider, and the service provider performs the work and returns a notice of completion during the same HTTP session. (Because the Java Message Service is inherently asynchronous, this chapter only discusses synchronous sessions configured to use HTTP for communication.)

If the proxy and service provider communicate asynchronously, that is, the proxy sends a request during one HTTP session, and the service provider responds send notice of completion of work during a later session, then you must perform additional tasks. For more information about asynchronous service providers, read this chapter, then see Chapter 4, "Asynchronous Service Provider Use Case."

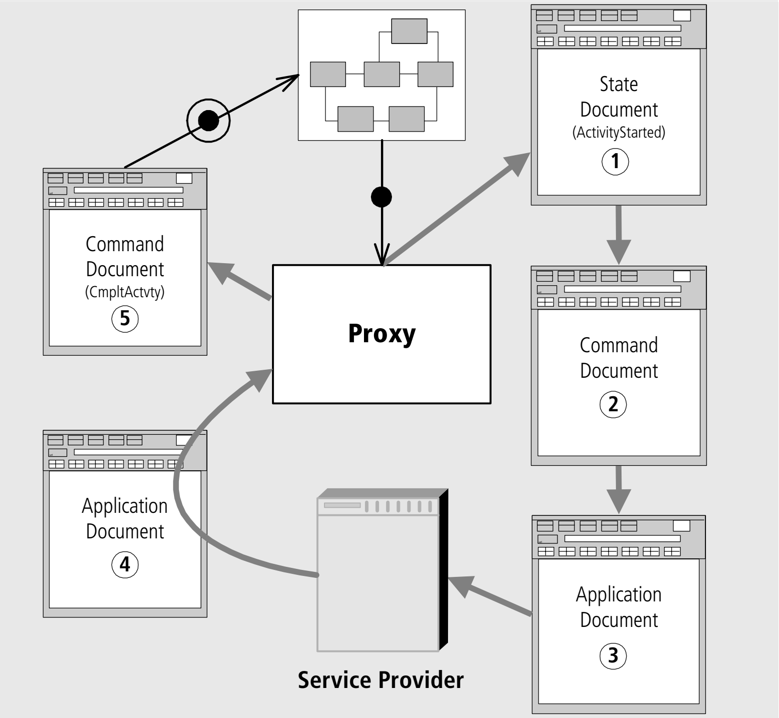
If the service provider requires authentication from the proxy before performing the activity, then you must perform additional tasks, also. For more information about enabling authentication for a service provider proxy, read this chapter (and Chapter 4 if applicable), then see Chapter 6, "Service Provider Authentication Use Case."

For information about the Fusion processing involved in requesting a service provider to perform an activity, see Appendix A, "Transforming Proxy Documents."

The service provider application (or its adapter) determines whether synchronous or asynchronous communication takes place with the partner proxy.

Use Case Summary

The following table provides an overview of this use case:

Use Case Information	Description
description	A proxy offers an ACTIVE activity to a service provider and waits for the service provider to send notification of completion of wok before the process can proceed.
expected outcome	The proxy sends a message to the application instructing it to start the activity; the application completes the activity and notifies the proxy
actors	▮ Fusion process engine ▮ application proxy ▮ service provider application
proxy document flow	**1** Proxy generates an "ActivityStarted" state document based on a call from the engine that the activity has entered the ACTIVE state. **2** Proxy generates command document with request for service provider. **3** Proxy sends HTTP request to service provider with application document in the message. **4** Service provider sends application document to proxy with message that activity was performed. **5** Proxy generates command document to notify engine that activity is complete.  State Document (ActivityStarted) 1 Command Document (CmpltActvty) 5 Proxy Command Document 2 Application Document 4 Application Document 3 Service Provider
required stylesheets	outbound stylesheet to: ▮ transform state document into command document that sends request to service provider (Step 2) ▮ generate application document containing request to service provider (Step 3) inbound stylesheet to transform application document into command document to notify engine that application performed the activity (Step 5)
other integrator tasks	▮ use application codes in the Fusion application dictionary that the application understands ▮ use the **AddAplUrl** FNscript command to configure the proxy as an HTTP client

The remainder of this chapter describes this use case in more detail and provides procedures for writing the stylesheets required for its successful completion.

Communicating Synchronously With a Service Provider

After a service requestor has started a Fusion process, the engine offers subsequent activities in the process to service provider applications. In the example application, after an order is placed, the engine asks the credit verification service to provide a credit check.

The process cannot proceed until the credit verification application has responded to its proxy with the results of the credit check. The proxy sends the request to the application as an HTTP request, and the service provider returns an acknowledgment of completion (or possibly inability to complete the activity) as the response for the HTTP request. The type of communication, in which the process waits for the application to respond before proceeding, is known as *synchronous* communication.

Because the request to the service provider and the application's response take place within the context of the same HTTP request/response session, your outbound stylesheet does not need to provide the application with identifying information about the activity, and the application does not need to return the identifying information to the proxy.

When the activity enters the ACTIVE state, the proxy generates a state document that:

- indicates that the proxy has started the activity
- contains the current value of any process attributes specified in the activity's application dictionary item
- contains identifying information about the activity, such as the unique process ID and activity ID that the engine generates, as well as the process name, activity name, and application code

The proxy uses the state document to generate a command document that:

- contains the FNCommand document element to identify itself as a command document
- sends a message to the application to perform the activity
- sends the application any values it needs to complete the activity

The proxy's outbound stylesheet must contain the rules that cause this command document to be generated correctly. The following subsections describe how to include these rules in your outbound stylesheet.

Configuration Note

To submit requests to the service provider, configure the proxy as an HTTP client. To accomplish this task, issue the **AddAplURL** FNscript command, specifying the application's network address to which requests should be sent.

For information about configuring proxies as clients and servers, see the *Forte Fusion Backbone System Guide*.

Step 1. Generating the Document Element

A proxy identifies a command document using the FNCommand element as its document element. The first task your outbound stylesheet must perform is to generate the document element for the command document.

For instructions for creating the FNCommand document element, see "Generating the Document Element" on page 46.

Step 2. Generating a Message to the Application

The proxy next must construct an application document to ask the application to perform the activity. To accomplish this task, your outbound stylesheet needs rules to:

- identify the started activity in the state document
- generate a command to send a message to the application
- create the FNMessage element whose content becomes the application document that is sent to the application
- generate the document element and any top-level child elements for the application document, so that the message is meaningful to the application

After you have created this application document structure, you can generate the content of the application document, as shown in the subsequent two sections.

To send a request to the service provider application, include a template like the following in your inbound stylesheet:

Match started activity in state doc
Send HTTP message
Specifies that app doc follows
Document element for app doc
Top-level element for app doc
Process children

```
<xsl:template match="FNCndState[@State='ActivityStarted']">
  <FNAplCommand Command="SendMessage" Method="Post"/>
    <FNMessage>
      <WorkRoot>
        <NewWork>
          <xsl:apply-templates/>
        </NewWork>
      </WorkRoot>
    </FNMessage>
  </FNAplCommand>
</xsl:template>
```

Transformation Notes

In the above template:

- WorkRoot and NewWork are shown as examples of a document element and top-level element, respectively, that an application might understand as indicating a new request from a proxy.
- The xsl:apply-templates element ensures that the child elements from the state document are processed to generate the content of the message, as shown in the next sections.

Step 3. Providing the Message Content

After you construct the application document, your stylesheet must include whatever information the application needs to perform the activity. In the example application, the service provider requires:

- the application code, which identifies the activity to the application

  The application code is defined in the application dictionary entry for the application when the Fusion process is defined. Thus the application should understand this value when it retrieves it from the application document.

- the list of process attributes, in a form that the application understands (described in "Sending Process Attribute Values" on page 60)

The application code value is included as one of the attributes of the FNIdentity element in the state document that the proxy generates for the activity.

To include the application code in the message to the application, include the following template in your outbound stylesheet:

Match FNIdentity in state doc
App doc element to hold app code
Copy value of application code

```
<xsl:template match="FNIdentity">
  <WorkType>
    <xsl:value-of select="@ActivityAppCode" />
  </WorkType>
</xsl:template>
```

Transformation Notes

The xsl:apply-templates element shown in the previous step ensures that the WorkType element is generated as a child of the NewWork element.

Step 4. Sending Process Attribute Values

You also must send the application the values of any process attributes that it uses. The state document that the proxy generates for the active activity contains an FNProcessAttributeList element containing FNProcessAttribute child elements. Each of the FNProcessAttribute elements corresponds to a process attribute in the application dictionary entry for the activity.

To send the process attribute values to the service provider application, your outbound stylesheet must transform these FNProcessAttributeList element and FNProcessAttribute elements in the state document into a form that the application can understand.

In the example application, values for two process attributes are sent to the CreditCheck service provider application:

- the original service requestor application document

  This document is the value of the StartingMessage process attribute, whose type is XmlData. This process attribute is transformed into an element called Order.

- a process attribute called CreditApproved, which holds the result of the credit check

  This process attribute is passed to the service provider as an element called CreditApproved. The process attribute is passed with a default value of “No,” and the application passes back an actual value (which also might be “No”) when it responds to the proxy after performing the credit check.

  For information on how the result value from the credit check is passed back to the proxy, see “Receiving Synchronous Notification of Completion” on page 64.

Note Because the service provider might send back a different value for this process attribute, it must have a lock type of Write or WriteQueue in the Fusion process definition. For information about creating process attributes and specifying lock types, see the *Forte Fusion Process Development Guide*.

For an example of a template that retrieves a value from a process attribute, see “Transforming Process Attribute Lists” on page 33. For an example of the transformations used to send such an application document to an application, see “Transmitting Application Documents as Process Attributes” on page 35.

Handling Redundant Values

In the example application, the information about the contents of the order is sent to the CreditCheck service provider application by simply passing the original application document. This method works fine in the example application, because the CreditCheck application (and the other service providers) uses the same XML vocabulary as the service requestor application.

In actual practice, your applications are likely to have different elements that correspond to Fusion process attributes. Thus, the outbound stylesheets for some proxies might need to generate individual elements to hold certain values, even though those values are embedded within the original service requestor application document.

Generated Document Examples

The following sections provide examples of the XML documents that a proxy generates when the activity for a service provider application enters the ACTIVE state.

State Document

A proxy generates a state document like the following after the Fusion process engine informs it that an activity has entered the ACTIVE state:

```
XML declaration                         <?xml version="1.0"?>

State doc document element              <FNState>
State of started activity                 <FNCndState State="ActivityStarted">
Process/activity information                <FNIdentity
                                                ProcessID="52"
                                                ProcessName="FNOrdersProcess"
                                                ActivityID="5"
                                                ActivityName="CreditCheck"
                                                ActivityAppCode="CreditCheck"/>
Process attribute list from app dict.       <FNProcessAttributeList>
Process attribute                             <FNProcessAttribute
                                                Name="CreditApproved"
                                                Type="TextData">
                                                No
                                              </FNProcessAttribute>
StartingMessage proc attribute                <FNProcessAttribute
                                                Name="StartingMessage"
                                                Type="XmlData">
Service requestor app document                <MsgDoc>
                                                <NewOrder>
                                                <Atts>
                                                <Att>
                                                  <AttName>Billee</AttName>
                                                  <AttType>XmlData</AttType>
                                                  <AttValue><![CDATA[fred]]></AttValue>
                                                </Att>
                                                <Att>
                                                  <AttName>Shippee</AttName>
                                                  <AttType>XmlData</AttType>
                                                  <AttValue><![CDATA[fred]]></AttValue>
                                                </Att>
                                                <Att>
                                                  <AttName>ItemCount</AttName>
                                                  <AttType>XmlData</AttType>
                                                  <AttValue><![CDATA[20]]></AttValue>
                                                </Att>
                                                <Att>
                                                  <AttName>OrderID</AttName>
                                                  <AttType>XmlData</AttType>
                                                  <AttValue><![CDATA[15]]></AttValue>
```

Process attribute

```
    </Att>
     </Atts>
     </NewOrder>
    </MsgDoc>
   </FNProcessAttribute>
   <FNProcessAttribute
    Name="Status"
    Type="TextData">
    Invoiced
   </FNProcessAttribute>
  </FNProcessAttributeList>
 </FNCndState>
</FNState>
```

Command Document

The following table shows the entire command document that is generated by the preceding steps.

Start command document
Command to send msg to app
Application document follows
Document element of app doc
App doc top-level element

Value of application code
Begin service provider value list
Service provider element to hold
original service requestor
application document

Start service requestor app doc

```
<FNCommand>
 <FNAplCommand Command="SendMessage" Method="Post">
  <FNMessage>
   <OrderEntered>
     <WorkRoot>
      <NewWork>
       <WorkType>CreditCheck</WorkType>
        <SPAtts>
          <SPAtt>
           <SPAttName>Order</SPAttName>
           <SPAttType>XmlData</SPAttType>
            <SPAttValue>
             <MsgDoc>
              <NewOrder>
               <Atts>
                <Att>
                  <AttName>Billee</AttName>
                  <AttType>TextData</AttType>
                  <AttValue>Jack</AttValue>
                </Att>
                <Att>
                  <AttName>Shippee</AttName>
                  <AttType>TextData</AttType>
                  <AttValue>Jill</AttValue>
                </Att>
                <Att>
                 <AttName>ItemCount</AttName>
                  <AttType>TextData</AttType>
                 <AttValue>12</AttValue>
                </Att>
                <Att>
```

```
                                  <AttName>OrderID</AttName>
                                  <AttType>TextData</AttType>
                                  <AttValue>555</AttValue>

                              </Atts>
                            </NewOrder>
```

End service requestor app doc

```
                          </MsgDoc>
                       </SPAttValue>
                       </SPAtt>
```

Service provider element to hold credit status value

```
                     <SPAtt>
                        <SPAttName>CreditApproved</SPAttName>
                        <SPAttType>TextData</SPAttType>
```

Default value for credit status

```
                        <SPAttValue>No</SPAttValue>
                     </SPAtt>
                   </SPAtts>
                 </NewWork>
             </WorkRoot>
      </OrderEntered>
```

Application doc completed

```
    </FNMessage>
  </FNAplCommand>
</FNCommand>
```

Application Document

The lines between the start and end tags of the FNMessage element in the above command document make up the application document that is sent to the service provider to perform the activity.

The actual command document that the example application generates has been modified slightly here to differentiate more clearly between the values from the service requestor (the Atts and Att elements in the original service requestor application document) and the values understood by the service provider (the SPAtts and SPAtt elements).

Receiving Synchronous Notification of Completion

After the service provider performs its activity, it sends an application document as an HTTP response to inform the proxy that the activity has been completed. This application document also contains the current values of any process attributes changed by the activity.

The proxy uses this application document to generate a command document that:

- contains the FNCommand document element to identify itself as a command document
- sends a command to the Fusion process engine to complete the activity
- returns values for any process attributes that were changed by the service provider

The proxy's inbound stylesheet must contain the rules that cause this command document to be generated correctly. The following subsections describe how to include these rules in your inbound stylesheet.

Step 1. Generating the Document Element

An application proxy identifies a command document by the FNCommand element as its document element. The first specific task your inbound stylesheet must perform is to generate the document element for the command document.

For instructions for creating the FNCommand document element, see "Generating the Document Element" on page 46.

Step 2. Sending a Command to the Engine

The command document next must instruct the engine to complete the activity. In the example application, the inbound stylesheet performs this task by matching the WorkCompleted element in the application document and transforming it into a CompleteActivity command.

To instruct the engine to complete the activity, include a template like the following in your inbound stylesheet:

Match WorkCompleted element in app doc
Send CompleteActivity command to engine
Process children

```
<xsl:template match="WorkCompleted">
  <FNCndCommand Command="CompleteActivity">
    <xsl:apply-templates/>
  </FNCndCommand>
</xsl:template>
```

Transformation Notes

In the above template:

- The "WorkCompleted" element from the application document indicates that the activity was successfully completed. If the application potentially could report some other result, you would need a template to match the result and generated the appropriate FNCndCommand Command attribute, such as "Rollback Activity" or "AbortActivity."
- The xsl:apply-templates element ensures that the process attribute list, to be generated by the next step, is placed within the start and end tags of the FNCndCommand element.

Step 3. Returning Updated Process Attribute Values

The final task the command document performs is to return the current values for any process attributes that were affected by the application. To accomplish this task, your inbound stylesheet must:

- construct an FNProcessAttributeList element
- construct an FNProcessAttribute element with the appropriate Name and Type attributes for each corresponding Att element (and its child elements) in the application document
- retrieve the current values from the application document

For information about transforming application document values into command document process attribute lists, see "Generating Process Attributes" on page 32.

Command Document Example

The following table shows the entire command document that is generated by the preceding steps.

```
Start command document               <FNCommand>
Command engine to complete activity    <FNCndCommand Command="CompleteActivity">
Start process attribute list             <FNProcessAttributeList>
App doc as process attribute               <FNProcessAttribute Name="StartingMessage"
                                                               Type="XmlData">
Doc element of app document                  <MsgDoc>
                                               <NewOrder>
                                                 <Atts>
                                                   <Att>
                                                      <AttName>Billee</AttName>
                                                      <AttType>TextData</AttType>
                                                      <AttValue>Jack</AttValue>
                                                   </Att>
                                                   <Att>
                                                      <AttName>Shippee</AttName>
                                                      <AttType>TextData</AttType>
                                                      <AttValue>Jill</AttValue>
                                                   </Att>
                                                   <Att>
                                                      <AttName>ItemCount</AttName>
                                                      <AttType>TextData</AttType>
                                                      <AttValue>12</AttValue>
                                                   </Att>
                                                   <Att>
                                                      <AttName>OrderID</AttName>
                                                      <AttType>TextData</AttType>
                                                      <AttValue>555</AttValue>
                                                   </Att>
                                                 </Atts>
                                               </NewOrder>
End of application document                  </MsgDoc>
                                           </FNProcessAttribute>
```

Updated process attribute

New process attribute value

End of process attribute list

```
      <FNProcessAttribute Name="CreditApproved"
                          Type="TextData">
    Yes
</FNProcessAttribute>
    </FNProcessAttributeList>
  </FNCndCommand>
</FNCommand>
```

Document Notes

In the above command document:

- Because there is no FNIdentity element to identify the process and activity, the proxy assumes that the document refers to the current process and activity.

  For information about command documents for asynchronous communication, see Chapter 4, "Asynchronous Service Provider Use Case."

- The CreditApproved process element contains the result of the application's credit check. This process attribute must have been defined in the application dictionary as writable.

Chapter 4

Asynchronous Service Provider Use Case

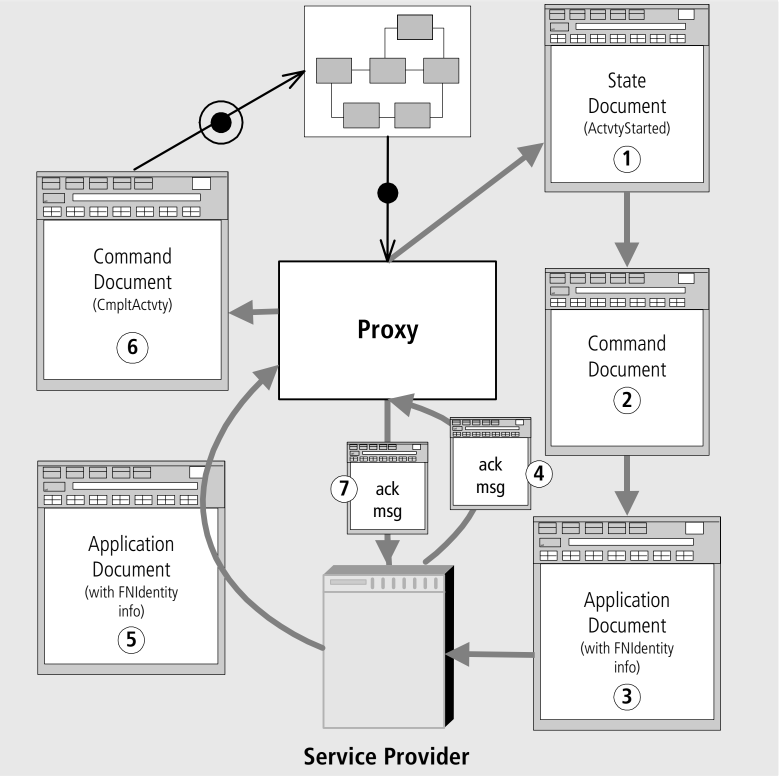
This chapter describes the XSL stylesheets and related integration tasks that enable a proxy to communicate asynchronously with a service provider application.

For basic information about submitting activities to service providers, see Chapter 3, “Synchronous Service Provider Use Case.”

The service provider application (or its adapter) determines whether synchronous or asynchronous communication takes place with the partner proxy. For asynchronous communication, proxies can be configured to use either HTTP or the Java Message Service (JMS).

Use Case Summary

The following table provides an overview of this use case:

Use Case Information	Description
description	A proxy offers an ACTIVE activity to a service provider, which returns the disposition of the request to the proxy at a later time, during a separate session.
expected outcome	The proxy sends a message to the application instructing it to start the activity; the application confirms receipt of the request, then performs the activity and notifies the proxy at a later time that it has completed the activity.
actors involved	▮ the Fusion process engine ▮ the application proxy ▮ the service provider application
proxy document flow	**1** Proxy generates an "ActivityStarted" state document based on a call from the engine that the activity has entered the ACTIVE state. **2** Proxy generates command document with request for service provider. **3** Proxy sends request to service provider as application document, which includes identifying information about the activity. **4** (HTTP sessions) Service provider acknowledges receipt of application document. **5** At some later point, service provider sends application document to proxy with completion message and information to identify the completed activity. **6** Proxy generates command document to notify engine that activity is complete. **7** (HTTP sessions) Proxy generates response to application to complete HTTP message/response communication  State Document (ActvtyStarted) 1 Command Document (CmpltActvty) 6 Proxy Command Document 2 7 ack msg ack msg 4 Application Document (with FNIdentity info) 5 Application Document (with FNIdentity info) 3 Service Provider

Use Case Information	Description
stylesheets required	outbound stylesheet to: ▮ transform state document into command document that sends request and identifying information to service provider (Step 2) ▮ generate application document containing request to service provider (Step 3) ▮ send acknowledgement to application's completion message (Step 7) inbound stylesheet to: ▮ receive (and ignore) notification that request from proxy was received (Step 4) ▮ transform subsequent application document into command document to notify engine that application performed the activity and identify the activity completed (Step 6)
Other integrator tasks	▮ use the **AddAplUrl** FNscript command to configure proxies as a sender (HTTP client or JMS sender) ▮ use the **AddListener** FNscript command to configure proxies as a listener (HTTP server or JMS listener) ▮ you can also use the **SetPort** FNscript command to configure the proxy as an HTTP server ▮ provide process definition with application codes that the application understands ▮ ensure that the partner application can store and return identifying information about the activity

The remainder of this chapter describes this use case in more detail and provides procedures for writing the stylesheets required for its successful completion. The example applies to proxies configured to use either HTTP or JMS messaging, calling out implementation differences between the communication protocols.

Communicating Asynchronously With a Service Provider

In some Fusion processes, a proxy might send an activity to a service provider as a request during one session, and the service provider responds with notice of completion of work during a separate session. Such behavior, in which other requests and responses intervene between the time the proxy sends the activity and the time the application sends notification of complication, is known as *asynchronous* communication.

For example, an order fulfillment process like the Fusion example application might include a service provider application that generates customer invoices. This application might receive requests from the proxy, then hold them and generate the invoices in batch mode at the end of the day. When the application notifies the proxy that an invoice has been generated, it must be able to identify the order to which the invoice refers.

When the service provider activity enters the ACTIVE state, the proxy generates a state document that:

- indicates that the proxy has started the activity
- contains the current value of any process attributes specified in the activity's application dictionary item
- contains identifying information about the activity, such as the process ID and activity ID, as well as the process name, activity name, and application code

Using the outbound stylesheet, the proxy uses the state document to generate a command document that:

- contains the FNCommand document element to identify itself as a command document
- sends a message to the application to perform the activity
- sends the application any process attribute values it needs to complete the activity
- sends the application the identifying information from the FNIdentity element

  When the application later responds that it has completed the activity, it returns any updated process attribute values, as well as the identifying information, so that the proxy knows which activity has been completed.

The proxy's outbound stylesheet must contain the rules that cause this command document to be generated correctly. The following subsections describes how to include these rules in your outbound stylesheet.

Also, your inbound stylesheet might need a template to process an intermediate application document that the service provider sends to acknowledge the proxy's request, before it responds with notification that the activity is completed. For information about writing this template, see "(HTTP Sessions) Receiving Acknowledgment from the Application" on page 75.

Configuration Notes

For asynchronous communication, a proxy must submit requests to the service provider during one session and receive acknowledgements during a separate session. Thus, you must configure the proxy as both a sender and listener (HTTP client/JMS sender and HTTP server/JMS listener).

Connecting to JMS provider application

For proxies configured to use JMS, use the **SetProtocol Name=JMS** Fusion Script command to specify information for connecting to a JMS provider application.

Configuring a Sender

Use the **AddAplURL** Fusion Script command to configure the proxy as an HTTP client or JMS sender. When issuing this command, for proxies configured for HTTP, specify the application's network address to which requests should be sent. For proxies configured for JMS, specify the messaging model (point-to-point or publish/subscribe) and other optional information for JMS messaging (such as JMS priority, persistence, and other JMS specifications).

Configuring a Listener

Use the **AddListener** Fusion Script command to configure the proxy as an HTTP server or JMS listener. When issuing this command specify the protocol you are using for messaging (HTTP or JMS).

If you specify HTTP, then you specify the port at which the proxy should receive messages from an application.

If specify JMS, then you specify the messaging model (point-to-point or publish/subscribe) and other optional information for JMS messaging (JMS message selectors, acknowledgement mode, and durable messaging behavior).

You can also use the **SetPort** command to configure an HTTP listener.

For more information about configuring proxies, see the *Forte Fusion Backbone System Guide*.

Step 1. Generating the Document Element

An application proxy identifies a command document by using the FNCommand element as its document element. The first specific task your inbound stylesheet must perform is to generate the document element for the command document.

For instructions for creating the FNCommand document element, see "Generating the Document Element" on page 46.

Step 2. Generating a Message to the Application

The proxy next must construct an application document to ask the application to perform the activity. To accomplish this task, your outbound stylesheet needs rules to:

- match the started activity in the state document
- generate a command to send a message to the application
- create the FNMessage element, whose content is the application document that is sent to the application
- generate the document element and any top-level child elements for the application document, so that the message is meaningful to the application

After you have created this application document structure, you can generate the content of the application document, as shown in subsequent sections.

To send a request to the service provider application, include a template similar to the following in your inbound stylesheet:

Match started activity in state doc

Send message

Specifies that app doc follows

Document element for app doc

Top-level element for app doc

Process children to hold app doc

```
<xsl:template match="FNCndState[@State='ActivityStarted']">

  <FNAplCommand Command="SendMessage" Method="Post">
    <FNMessage>
      <WorkRoot>
        <NewWork>
          <xsl:apply-templates/>
        </NewWork>
      </WorkRoot>
    </FNMessage>
  </FNAplCommand>
</xsl:template>
```

Transformation Notes

In the above template:

- WorkRoot and NewWork are shown as examples of elements that an application might understand as indicating a new request from a proxy.
- The xsl:apply-templates element ensures that the child elements from the state document, that is, the activity information and process attributes, are processed to generate the content of the message, as shown in the next sections.
- If your proxy is configured to use JMS, you must still specify an HTTP method for the FNAplCommand even though this method is ignored when using JMS.

Step 3. Providing Identifying Information

After you construct the application document, your stylesheet must generate its content. In the case of asynchronous processing, the application document must include the process ID and activity ID from the state document. These two values together uniquely identify the activity to the Fusion process engine. The application must store these values until it performs the activity, and it then must return them later to the proxy to identify the activity that was performed.

The application also can include the application code, which is included in the state document. This value, which identifies the activity for the application to perform, is defined in the application dictionary entry when the Fusion process is defined.

You also must provide the application with any process attributes it requires, in a form that the application understands. For information about transforming process attributes, see “Sending Process Attribute Values” on page 74.

The application code, process ID, and activity ID are contained in the state document as values of attributes of the FNIdentity element. The service provider uses the application code to determine what activity to perform. For more information about passing the application code to the service provider, see “Providing the Message Content” on page 59.

In most cases, however, the application does not use the process ID and activity ID as it does the application code, but simply stores these values as a string to pass back to the proxy after performing the activity. Because the application does not care about the contents of this string, the simplest method is pass the entire FNIdentity element to the application, even though it contains redundant or unnecessary information, such as the ActivityName attribute.

Depending on the application, your outbound stylesheet might be able simply to copy the FNIdentity element as it appears in the state document. Or you might need to place the FNIdentity element inside an element that is part of the application's vocabulary, for example, an "AddtlInfo" element that the application uses to store external information.

To summarize the transformations you must include in your outbound stylesheet:

1. Match the FNIdentity element in the state document.
2. Transform the value of the application code attribute into an element that the application can use.
3. Pass the entire FNIdentity element to the application, possibly placing it inside an element that is part of the application's XML vocabulary.

To provide this information in the message to the application, include a template like the following in your outbound stylesheet:

Match FNIdentity in state doc
App doc element to hold app code
Copy value of application code

Create element to hold FNIdentity
Copy FNIdentity and its attributes

```
<xsl:template match="FNIdentity">
  <WorkType>
    <xsl:value-of select="@ActivityAppCode"/>
  </WorkType>
  <AddtlInfo>
    <xsl:copy-of select="."/>
  </AddtlInfo>
</xsl:template>
```

Transformation Notes

In the above templates:

- The xsl:copy-of element copies the FNIdentity element and everything below it in the source document tree, which in this case are the attributes of FNIdentity.
- The "." expression indicates the current node in the source document, that is, the point at which the match in the xsl:template element occurred (which in this case is FNIdentity).
- If you pass the FNIdentity element to the application without placing it within another element, you can omit the <AddtlInfo> and </AddtlInfo> lines.

Step 4. Sending Process Attribute Values

You also must send the application the values of any process attributes that it uses. The state document that the proxy generates for the active activity contains an FNProcessAttributeList element containing FNProcessAttribute child elements. Each of these FNProcessAttribute elements corresponds to a process attribute in the application dictionary entry for the activity.

To send the process attribute values to the service provider application, your outbound stylesheet must transform the FNProcessAttributeList element and FNProcessAttribute elements in the state document into a form that the application understands.

This process is identical for synchronous and asynchronous activities. In the example application (which contains no asynchronous activities), values for two process attributes are sent to the CreditCheck service provider application:

- the original service requestor application document

  This document is the value of the StartingMessage process attribute, whose type is XmlData. This process attribute is transformed into an element called Order.

- a process attribute called Status, which holds the current status of the order

  When the billing application notifies the proxy that it has performed its activity, the application document it sends back changes the status of the order to “Invoiced.”

  For information on how the result value from the credit check is passed back to the proxy, see “Returning Updated Process Attribute Values” on page 83.

Note Any process attribute for which the service provider might send back an updated value must be specified with a lock type of Write or Write Queue in the Fusion process definition. For information about creating process attributes, see the *Forte Fusion Process Development Guide*.

For an example of a template that retrieves a value from a process attribute, see “Transforming Process Attribute Lists” on page 33. For an example of the transformations used to send an entire application document as a single value, see “Transmitting Application Documents as Process Attributes” on page 35.

Step 5. (HTTP Sessions) Receiving Acknowledgment from the Application

Because every HTTP request requires a response, the service provider must acknowledge that it has received the proxy's request to perform the activity. This response is necessary because the application does not send the disposition of the activity until some later time, at which point a new request/response cycle is initiated.

Note This second request/response cycle must be completed by a subsequent acknowledgment by the proxy. For information about handling the application document received after the activity is performed, see "Receiving Asynchronous Notification of Completion" on page 80.

The service provider can respond in either of two ways to the proxy's request to perform the activity:

- send only an HTTP header to complete the cycle

  In this case, there is no message content for your inbound stylesheet to process.

- send an application document containing the acknowledgment

  To avoid any unnecessary or possibly confusing processing of the application document, it is good practice to use an empty template to ensure that the proxy ignores this message. This template must match whatever application document element specifies the acknowledgment.

  You include this template in your inbound stylesheet for the application's proxy, because the communication is going from the application to the proxy.

To ensure that the proxy ignores the acknowledgment message, include a template like the following in your inbound stylesheet:

Match status attr in app doc

```
<xsl:template match="WorkStatus[@Status='Received']"/>
```

Transformation Notes

In the above template:

- The inbound stylesheet matches the WorkStatus element whose Status attribute has a value of "Received," then instructs the XSL processor to ignore it.
- This example is an arbitrary representation of how an application might send the acknowledgment of the proxy's request. The actual template your inbound stylesheet uses depends, of course, on the XML your application uses to represent the acknowledgment.

Generated Document Examples

The following sections provide examples of the XML documents that a proxy generates when the activity for a service provider application enters the ACTIVE state.

State Document

A proxy generates a state document like the following after the Fusion process engine informs it that an activity has entered the ACTIVE state:

```
XML declaration                         <?xml version="1.0"?>
State doc document element              <FNState>
State of started activity                 <FNCndState State="ActivityStarted">
Process/activity information                <FNIdentity
                                                ProcessID="52"
                                                  ProcessName="FNOrdersProcess"
                                                  ActivityID="5"
                                                  ActivityName="OrderVerification"
                                                  ActivityAppCode="OrderVerification"/>
Process attribute list from app dict.       <FNProcessAttributeList>
Status process attribute                      <FNProcessAttribute Name="Status" Type="TextData">
current value from prev. activity               CreditVerified
                                              </FNProcessAttribute>
StartingMessage proc attribute                <FNProcessAttribute Name="StartingMessage"
                                                                   Type="TextData">
Service requestor app document                  <MsgDoc>
                                                  <NewOrder>
                                                    <Atts>
                                                        <Att>
                                                           <AttName>Billee</AttName>
                                                           <AttType>TextData</AttType>
                                                        <AttValue><![CDATA[fred]]></AttValue>
                                                     </Att>
                                                     <Att>
                                                        <AttName>Shippee</AttName>
                                                        <AttType>TextData</AttType>
                                                        <AttValue><![CDATA[fred]]></AttValue>
                                                     </Att>
                                                     <Att>
                                                        <AttName>ItemCount</AttName>
                                                        <AttType>TextData</AttType>
                                                        <AttValue><![CDATA[20]]></AttValue>
                                                     </Att>
                                                     <Att>
                                                        <AttName>OrderID</AttName>
                                                        <AttType>TextData</AttType>
                                                        <AttValue><![CDATA[15]]></AttValue>
                                                     </Att>
                                                    </Atts>
                                                  </NewOrder>
                                                </MsgDoc>
```

```
      </FNProcessAttribute>
    </FNProcessAttributeList>
  </FNCndState>
</FNState>
```

Document Notes

The above state document is identical to the one shown in the synchronous example in Chapter 3, “Synchronous Service Provider Use Case.” The engine always provides the same information about the activity, and the proxy includes this information in the state document, whether the activity is synchronous or asynchronous.

Command Document

The following command document to send the activity request to the service provider is generated by the preceding steps.

```
Start command document              <FNCommand>
Command to send msg to app            <FNAplCommand Command="SendMessage" Method="Post">
Application document follows            <FNMessage>
Document element of app doc               <OrderEntered>
App doc top-level element                     <WorkRoot>
                                                <NewWork>
Value of application code                         <WorkType>CreditCheck</WorkType>
Element to hold FNIdentity                          <AddtlInfo>
Copy FNIdentity from state doc                         <FNIdentity>
                                                          ProcessID="52"
                                                          ProcessName="FNOrdersProcess"
                                                          ActivityID="5"
                                                          ActivityName="BillPreparation"
                                                          ActivityAppCode="SendInvoice"/>
                                                     </AddtlInfo>
Begin service provider value list                    <SPAtts>
Service provider element to hold                        <SPAtt>
   original service requestor                              <SPAttName>Order</SPAttName>
   application document                                    <SPAttType>XmlData</SPAttType>
                                                             <SPAttValue>
Start service requestor app doc                               <MsgDoc>
                                                                 <NewOrder>
                                                                    <Atts>
                                                                       <Att>
                                                                          <AttName>Billee</AttName>
                                                                            <AttType>TextData
                                                                            </AttType>
                                                                          <AttValue>Jack</AttValue>
                                                                       </Att>
                                                                       <Att>
                                                                           <AttName>Shippee
                                                                             </AttName>
                                                                           <AttType>TextData
                                                                             </AttType>
```

```
                                      <AttValue>Jill
                                        </AttValue>
                                  </Att>
                                  <Att>
                                     <AttName>ItemCount
                                       </AttName>
                                     <AttType>TextData
                                       </AttType>
                                     <AttValue>12</AttValue>
                                  </Att>
                                  <Att>
                                     <AttName>OrderID
                                       </AttName>
                                     <AttType>TextData
                                       </AttType>
                                     <AttValue>555</AttValue>
                                  </Att>
                                  </Atts>
                               </NewOrder>
```

End service requestor app doc

```
                            </MsgDoc>
                          </SPAtt>
```

Service provider element to hold billing status value

```
                     <SPAtt>
                        <SPAttName>Invoiced</SPAttName>
                        <SPAttType>TextData</SPAttType>
```

Default value for invoice status

```
                        <SPAttValue>No</SPAttValue>
                     </SPAtt>
                   </SPAtts>
                 </NewWork>
             </WorkRoot>
        </OrderEntered>
```

Application doc completed

```
    </FNMessage>
  </FNAplCommand>
</FNCommand>
```

Document Notes

The above command document is largely the same as that shown on page 62 for a synchronous service provider activity, except that the asynchronous version adds the AddtlInfo element to hold the FNIdentity element and its attributes.

Application Document

The lines between the start and end tags of the FNMessage element in the above command document make up the application document that is sent to the service provider to request that it perform the activity.

In this application document:

- The values from the service requestor (the Atts and Att elements in the original service requestor application document) are distinguished from the values understood by the service provider (the SPAtts and SPAtt elements).
- The FNIdentity elements and attributes (contained within the AddtlInfo element) are included to identify the activity for asynchronous processing.
- The service provider might return another application document to acknowledge the proxy's request. For information on handling such a document, see "(HTTP Sessions) Receiving Acknowledgment from the Application" on page 75.

Receiving Asynchronous Notification of Completion

After an asynchronous service provider performs the activity requested, it returns an application document to its proxy. This document contains:

- a message that the activity was performed
- information to identify the activity performed

The proxy transforms this application document into a command document that tells the Fusion process engine to complete the activity. The proxy's inbound stylesheet must contain the rules that cause this command document to be generated correctly.

HTTP sessions

For HTTP sessions, because the application document begins a new HTTP request/response cycle, the command document must contain an instruction to send a response to the application, thus completing the cycle.

The following subsections describes how to include the necessary templates in your inbound stylesheet to accomplish these tasks.

Step 1. Generating the Document Element

An application proxy identifies a command document by using the FNCommand element as its document element. The first specific task your inbound stylesheet must perform is to generate the document element for the command document.

For instructions for creating the FNCommand document element, see "Generating the Document Element" on page 46.

Step 2. Sending a Command to the Engine

The command document next must instruct the engine to complete the activity. In the example application, the inbound stylesheet performs this task by matching the WorkCompleted element in the application document and transforming it into a CompleteActivity command.

Note Although the service provider activities in the example application are all synchronous, the command to the engine is the same for asynchronous activities. The only difference in the command document is the FNIdentity element, as described in "Identifying the Completed Activity to the Proxy" on page 82.

To instruct the engine to complete the activity, include a template like the following in your inbound stylesheet:

Match WorkCompleted element in app doc
Send CompleteActivity command to engine
Process children

See page 84 for acknowledgment command

```
<xsl:template match="WorkCompleted">
  <FNCndCommand Command="CompleteActivity">
    <xsl:apply-templates/>
  </FNCndCommand>
acknowledgment to application
</xsl:template>
```

Transformation Notes

In the above template:

- The "WorkCompleted" element from the application document indicates that the activity was successfully completed. If the application reported some other result, you would need a template that matched the result and generated the appropriate FNCndCommand Command attribute, such as "RollbackActivity" or "AbortActivity."
- The xsl:apply-templates element ensures that the FNIdentity element and the process attribute list, which are generated by the templates shown in the next two sections, respectively, are placed within the start and end tags of the FNCndCommand element.
- The xsl:template element that matches the completed activity contains both the FNCndCommand element to send the completion command to the engine and the FNAplCommand element to send acknowledgment of completion to the application.

  For more information about acknowledging the completion of the activity, see "(HTTP Sessions) Acknowledging the Completion Message" on page 84.

Step 3. Identifying the Completed Activity to the Proxy

The next task your inbound stylesheet must perform is to return the process ID and activity ID to the proxy, so that the proxy knows which activity to complete. Assuming that your outbound stylesheet passed the entire FNIdentity element as described in “Providing Identifying Information” on page 72, your inbound stylesheet can retrieve this information from the application document in either of two ways.

FNIdentity as child element

If the FNIdentity element was passed inside another element, provide a template that matches that element, then copies into the command document the FNIdentity element inside of the parent element.

To retrieve the FNIdentity element from inside another element, include a template like the following in your inbound stylesheet:

Match AddtlInfo element in app doc
Copy FNIdentity element inside it

```
<xsl:template match="AddtlInfo">
  <xsl:copy-of select="FNIdentity"/>
</xsl:template>
```

FNIdentity by itself

If the FNIdentity element was passed on its own, provide a template that matches it and copies it into the command document.

To retrieve the FNIdentity element directly from the application document, include the following template in your inbound stylesheet:

Match FNIdentity element in app doc
Copy it to command doc

```
<xsl:template match="FNIdentity">
  <xsl:copy-of select="."/>
</xsl:template>
```

Transformation Notes

In the above templates:

- Because the xsl:copy-of element is by definition recursive, copying the entire segment of the source node tree from the point of the match, there is no need for an xsl:apply-templates element to ensure that any children of the FNIdentity element are copied.
- The “.” expression in the second template above indicates that the copying should start at the point where the match was found, that is, at the FNIdentity element itself.
- The proxy only requires the process ID and activity ID attributes to identify the activity to complete. However, because the proxy ignores the additional information, it is simpler to return the entire FNIdentity element than to attempt to extract the two attributes from it.

Step 4. Returning Updated Process Attribute Values

The final task the command document performs is to return the current values for any process attributes that were affected by the application. In the example application, the billing service provider returns an updated value of the Status process attribute to indicate that it has sent an invoice for this order.

Note The application also returns the original service requestor application document as the value of the StartingMessage process attribute. However, the application does not change the value of this process attribute.

To return process attribute values to the proxy, your inbound stylesheet must:

- construct an FNProcessAttributeList element
- construct an FNProcessAttribute element with the appropriate Name and Type attributes for each corresponding element (and its child elements) in the application document
- retrieve the current values from the application document

For information about transforming application document values into command document process attribute lists, see “Generating Process Attributes” on page 32.

Step 5. (HTTP Sessions) Acknowledging the Completion Message

When the service provider sends the application document containing the completion information, it initiates a new HTTP request/response cycle. Therefore, the command document that the proxy generates must send a message back to the application to complete the cycle.

The simplest way to accomplish this task is to generate an empty response message to the application. You can place this message within the template that matches the completed activity in the application document, as shown in "Sending a Command to the Engine" on page 81. Because you are generating the command to the engine and the command to the application as a result of matching the same element in the application document, your stylesheet only needs one template to accomplish both tasks.

To generate a response to the service provider, include the following element in your inbound stylesheet within the template that matches the completed activity in the application document:

Generate empty msg command

```
<FNAplCommand Command="SendResponse"/>
```

Transformation Notes

In the above template:

- The empty FNAplCommand element causes an HTTP message to be sent with a header and no content. This communication is sufficient to complete the request/response cycle.
- The empty FNAplCommand is an acceptable response to an incoming JMS message (although not required when using JMS). This means you can use the same template to handle incoming HTTP and JMS messages.
- For the complete template of which the application command is part, see "Sending a Command to the Engine" on page 81.

Command Document Example

The following command document is generated by the preceding steps.

```
Start command document                <FNCommand>
Command engine to complete activity     <FNCndCommand Command="CompleteActivity">
FNIdentity element returned by app        <FNIdentity
                                              ProcessID="52"
                                              ProcessName="FNOrdersProcess"
                                              ActivityID="5"
                                              ActivityName="BillPreparation"
                                              ActivityAppCode="SendInvoice"/>
Start process attribute list              <FNProcessAttributeList>
App doc as process attribute                <FNProcessAttribute Name="StartingMessage"
                                                                Type="XmlData">
Doc element of app document                   <MsgDoc>
                                                <NewOrder>
                                                  <Atts>
                                                    <Att>
                                                       <AttName>Billee</AttName>
                                                       <AttType>TextData</AttType>
                                                       <AttValue>Jack</AttValue>
                                                    </Att>
                                                    <Att>
                                                       <AttName>Shippee</AttName>
                                                       <AttType>TextData</AttType>
                                                       <AttValue>Jill</AttValue>
                                                    </Att>
                                                    <Att>
                                                       <AttName>ItemCount</AttName>
                                                       <AttType>TextData</AttType>
                                                       <AttValue>12</AttValue>
                                                    </Att>
                                                    <Att>
                                                       <AttName>OrderID</AttName>
                                                       <AttType>TextData</AttType>
                                                       <AttValue>555</AttValue>
                                                    </Att>
                                                  </Atts>
                                                </NewOrder>
End of application document                   </MsgDoc>
                                            </FNProcessAttribute>
Updated process attribute                   <FNProcessAttribute Name="Status"
                                                                Type="TextData">
New process attribute value                   Invoiced
                                            </FNProcessAttribute>
End of process attribute list             </FNProcessAttributeList>
                                        </FNCndCommand>
App command to send empty response      <FNAplCommand Command="SendResponse"/>
                                      </FNCommand>
```

Document Notes

In the above document:

- There are two children of the FNCommand document element—the FNCndCommand element to send the CompleteActivity instruction to the engine, and the FNAplCommand to send the empty response to the service provider.
- The entire FNIdentity element is copied directly from the application document. The proxy ignores any attributes it does not need.
- The value of the Status process attribute is changed to “Invoiced” to reflect the status of the order as returned by the billing application.

Chapter 5

Service Requestor Authentication Use Case

This chapter describes how to enable a service requestor application to authenticate itself to a proxy (configured for HTTP) to initiate a Fusion process. The chapter discusses the authentication documents you might need to write and related integration tasks you need to perform. Authentication discussed in this chapter applies only to proxies configured for HTTP communication.

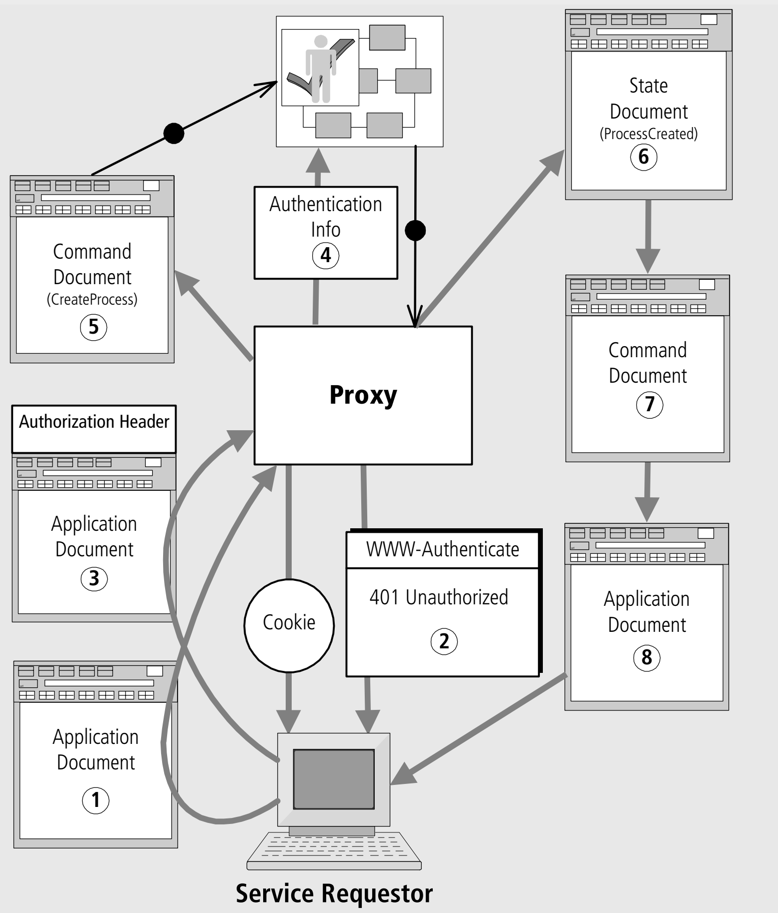
For basic information about submitting a request to a proxy to create a Fusion process, see Chapter 2, "Service Requestor Use Case."

For general information about authentication in a Fusion enterprise application (including authentication for proxies configured for JMS), see the *Forte Fusion Backbone System Guide*.

Use Case Summary

The following table provides an overview of this use case:

Use Case Information	Description
description	A service requestor application attempts to initiate a Fusion process with a proxy that requires authentication.
expected outcome	The application's proxy accepts the application's authentication, instructs the Fusion engine to start the process, and notifies the application that the process has been started.
actors involved	▮ the service requestor application ▮ the application proxy ▮ the Fusion process engine
proxy document and message flow	**1** Service requestor sends application document to proxy. **2** Proxy responds with message containing HTTP 401 Unauthorized error and WWW-Authenticate header specifying the type of authentication it requires (Basic or FusionXML). **3** Service requestor resends application document with authorization header containing authentication information (either user name/password combination or XML authentication document). **4** Proxy submits the authentication information to the Fusion engine as a user profi.le. **5** After the engine validates the user profile, the proxy submits a command document to create the process. **6** Proxy creates state document based on information received from engine about the process. **7** Proxy generates command document that includes message for application that request was received. **8** Proxy sends: ▮ application document to service requestor ▮ HTTP cookie to service requestor to authenticate it for rest of current session

Use Case Information	Description
	 State Document (ProcessCreated) 6 Authentication Info 4 Command Document (CreateProcess) 5 Command Document 7 Proxy Authorization Header Application Document 3 WWW-Authenticate 401 Unauthorized 2 Cookie Application Document 8 Application Document 1 Service Requestor
stylesheets required	inbound stylesheet to transform application document into command document to start process (Step 5) outbound stylesheet to: ▮ transform state document into command document to notify application (Step 6) ▮ generate application document containing notification (Step 8) No stylesheets are required to process an authentication document (for FusionXML authentication), because you create this document using Fusion’s XML structure and vocabulary.
Other integration tasks	▮ use the **SetAuthentication** FNscript command to configure the proxy to require authentication and to specify the type of authentication ▮ create a Fusion user validation against which to authentication the user information submitted by the service requestor

The remainder of this chapter describes this use case in more detail and provides procedures for writing authentication documents and performing the tasks required for authentication.

Authenticating a Service Requestor with a Proxy

When a service requestor sends its proxy a request to start a Fusion process, the proxy may require authentication from the service requestor before instructing the Fusion process engine to create the process.

Authentication message flow

The following flow of documents and messages takes place when a service requestor attempts to start a Fusion process with a proxy that requires authentication:

1. The service requestor sends an application document to the proxy.
2. The proxy rejects the document, and returns a “401 Unauthorized” error message with a WWW-Authenticate header specifying the authentication type (Basic or FusionXML).
3. The service requestor resubmits the application document with the appropriate Fusion user information included in the authorization header of the HTTP message.

   The specific information you supply depends on whether the proxy requires Basic or FusionXML authentication. For information about supplying user validation information, see “Submitting Authentication Information” on page 93.

Note You could submit the user information with the application document from the beginning and bypass Step 2 and Step 3. However, it is good practice for security reasons not to send the authorization information until the proxy requests it.

4. The proxy submits this user information to the Fusion process engine, which validates it against the user validation you specified when you created the Fusion process definition.

   For more information about Fusion user validations, see the *Forte Fusion Process Development Guide*.
5. If the engine validates the user information that the proxy submitted on behalf of the service requestor, the proxy:
   - creates an HTTP session for the service requestor
   - sends a cookie to the service requestor, so that the application is authorized for subsequent requests during the current HTTP session
6. The proxy now can submit a command document to the engine requesting that it create the process for the service requestor.

When the proxy accepts the application document from the service requestor, the procedure for creating the process and notifying the application is the same as described in Chapter 2, “Service Requestor Use Case.”

Enabling authentication

To enable the service requestor to be authenticated and the process created, you must take the following steps:

1. Configure the proxy to require authentication, and specify the type of configuration required.
2. Create a Fusion user validation against which to validate the user information that the proxy submits.
3. Map your service requestor application users to users recognized by the Fusion user validation.
4. Configure the proxy with the proper identification corresponding to the Fusion user validation.

The following sections describe how to perform these tasks.

Step 1. Configuring the Proxy for Authentication

To require authentication from a service requestor, you must configure its proxy for the type of authentication desired. To do so, issue the **SetAuthentication** FNScript command as follows:

Basic authentication

```
fnscript> SetAuthentication Basic
```

With Basic authentication, the service requestor sends a user name and password combination to the proxy. For information about supplying this information, see "Sending a User Name and Password" on page 93.

FusionXML authentication

```
fnscript> SetAuthentication FusionXML
```

With FusionXML authentication, the service requestor sends an XML authentication document with additional user information, for example, a Fusion user profile or role. For more information about creating authentication documents, see "Sending an Authentication Document" on page 93.

Setting Session Parameters

You also can use the following FNscript commands to specify how the proxy handles user sessions:

Command	Description
SetSessionMaximum	specifies how many concurrent sessions the proxy accepts from the service requestor
SetSessionTimeout	specifies how long the proxy waits for a response from a service requestor
SetAuthentication Server=Local | Process Engine	specifies the source of the user authentication information

For more information about proxy configuration, see the *Forte Fusion Backbone System Guide*.

Step 2. Creating a User Validation

When you define a Fusion process, you create a user validation to authorize connections to the process engine. The user validation examines information about the user to determine whether or not the user is authorized to connect to a Fusion engine. Any application that attempts to start a session with the engine is verified by the user validation.

If the service requestor's proxy is configured to require authentication from its partner application, then the proxy passes any user information that the application submits to it (through either a user/password combination or an authentication document) to the engine to verify against the user validation.

For information about submitting application user information to the proxy, see "Submitting Authentication Information" on page 93.

For information about defining Fusion user validations, roles, and user profiles, see the *Forte Fusion Process Development Guide* and the online Help for the appropriate Workshops.

Step 3. Mapping Application Users to Fusion Users

For either Basic or FusionXML authentication, the proxy takes the information the application supplies and submits it to the Fusion process engine for validation. Neither the proxy nor the engine has any knowledge of the application user. Therefore, your application must be able to map its users to a Fusion user recognized by the user validation defined for the engine.

The simplest way for an application to supply the correct user information to the proxy is for it to map all valid users to a small set of Fusion users, or even a single Fusion user. For example, you might create a Fusion user called OrderClerk. When both Mary Smith and Joe Young connect to the service requestor application, the application passes the OrderClerk user name and password to the proxy to authenticate against the user validation for the process.

If you are using FusionXML authentication to specify additional information, the same concept applies to user roles and profiles. That is, the application must map each of its authorized users to a Fusion role or profile as required by the user validation for the engine.

For example, the Fusion process that the example Fusion application uses contains a role called ProcessCreator. If the example application required FusionXML authentication, then the StartAndVerify service requestor application would need to pass this role name each time an application user attempted to start a session with the proxy.

Step 4. Submitting Authentication Information

When a proxy that requires authentication returns a 401 Unauthorized message, you must resubmit the service requestor application document with the appropriate authorization information in the message header. The type of authorization you specified using the FNscript proxy configuration command **SetAuthentication** determines the information you must include:

- For Basic authentication, include a *user name:password* string.
- For FusionXML authentication, include a string containing an XML authentication document.

The following subsections describe how to use each of these methods.

For information about configuring proxies, see "Configuring the Proxy for Authentication" on page 91.

Sending a User Name and Password

If the proxy uses Basic authentication, include the following string in the authorization header for the application document:

```
Authorization: basic base64(name:password)
```

where *name* and *password* are a Fusion user name and password accepted by the user validation for the Fusion process.

For example, assume that user name "OrderClerk" and password "$$clerk##" are a valid Fusion user name and password for this process. When the service requestor is prompted by the proxy to submit authentication, you would include the following string in the HTTP message header:

```
Authorization: basic base64('orderclerk:$$clerk##')
```

The name and password are encoded with Base64 and sent to the proxy, which submits the information to the engine. If the Fusion engine validates this user name and password, the proxy processes the command document instructing the engine to create the process.

Sending an Authentication Document

If the proxy uses FusionXML as its authentication type, then the service requestor submits an *authentication document* in its response to the proxy's WWW-Authenticate message. The authentication document can contain additional Fusion user information beyond the name and password, such as roles and user profiles.

To send an authentication document to a proxy, include the following string in the authorization header for the application document:

```
Authorization: FusionXML base64(XML_authentication_document)
```

where *XML_authentication_document* is a string containing an XML document. For information about specifying the contents of this document, see the following section, "Creating an Authentication Document."

If the Fusion engine validates the contents of the authentication document to the proxy, the proxy then submits the command document instructing the engine to create the process.

Creating an Authentication Document

An authentication document is an XML document that uses a specific structure and vocabulary. For information about the format of an authentication document, see Appendix B, “Proxy Document Element Hierarchies” and the Fusion Backbone online Help.

The FNAuthenticate element is required, because it is the document element that identifies the string as an authentication document. Although, strictly speaking, all other elements are optional, you must include all values required by the Fusion user validation, for example, a user profile or role.

When you create the document, you use the XML structure and vocabulary that Fusion understands, so no transformations are required. There are no stylesheets associated with authentication documents, and the proxy’s XSL processor never processes them.

Example Authentication Document

The following example shows an authentication document that a service requestor might submit to a proxy:

Document element

Fusion user profile

Fusion user information

List of Fusion roles

Fusion roles from proxy configuration

```
<FNAuthenticate>
  <FNUserProfile Name="FNProxyProfile"/>
    <FNUser Name="ProxyUser"
              Password="pxy$usr"
              OtherInfo="telesales">
  <FNRoleList>
    <FNRole Name="ProcessCreator"/>
    <FNRole Name="OrderTaker"/>
  </FNRoleList>
</FNAuthenticate>
```

Chapter 6

Service Provider Authentication Use Case

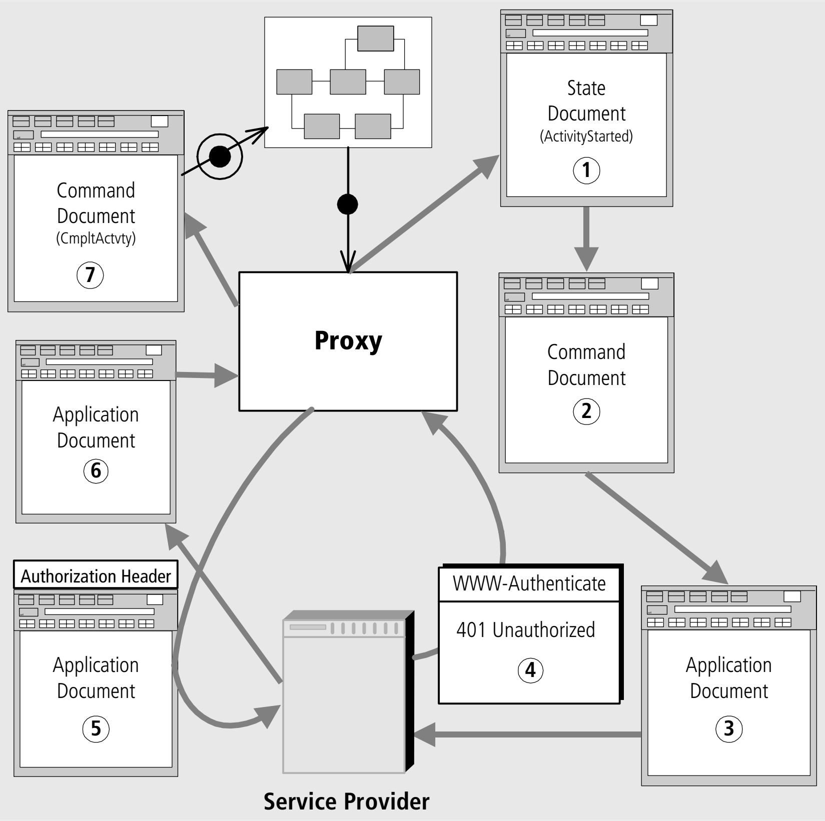
This chapter describes the integration tasks required to enable a proxy (configured for HTTP) to authenticate itself with a service provider application. Authentication discussed in this chapter applies only to proxies configured for HTTP communication.

For information about the basic tasks and stylesheets required when a proxy submits an activity to a service provider, see Chapter 3, “Synchronous Service Provider Use Case” or Chapter 4, “Asynchronous Service Provider Use Case.”

For general information about authentication in a Fusion enterprise application (including authentication for proxies configured for JMS), see the *Forte Fusion Backbone System Guide*.

Use Case Summary

The following table provides an overview of this use case:

Use Case Information	Description
description	A proxy submits a request to perform an activity to a service provider application that requires authentication.
expected outcome	The application accepts the proxy's authentication, performs the activity, and notifies the proxy when it is completed.
actors involved	▮ the application proxy ▮ the service requestor application (the Fusion process engine is not involved in the authentication process, only in submitting the activity to the proxy, then completing it when notified by the proxy)
proxy document and message flow	**1** Proxy generates an "ActivityStarted" state document based on a call from the engine that the activity has entered the ACTIVE state. **2** Proxy generates command document with request for service provider. **3** Proxy sends request to service provider as application document. **4** Service provider responds with message containing HTTP 401 Unauthorized error and WWW-Authenticate header specifying the type of authentication it requires (Basic or FusionXML). **5** Proxy resends application document with authorization header containing authentication information (either user name/password combination or XML authentication document). **6** Service provider performs activity and sends application document to proxy with message that activity was performed. **7** Proxy generates command document to notify engine that activity is complete
	 State Document (ActivityStarted) 1 Command Document (CmpltActvty) 7 Proxy Command Document 2 Application Document 6 Authorization Header Application Document 5 WWW-Authenticate 401 Unauthorized 4 Application Document 3 Service Provider

Use Case Information	Description
stylesheets required	outbound stylesheet to: - transform state document into command document that sends request to service provider (Step 2) - generate application document containing request to service provider (Step 3) inbound stylesheet to transform application document into command document to notify engine that application performed the activity (Step 7) No stylesheets are required for the proxy to create an authentication document, because the proxy already knows the required XML structure and vocabulary.
Other integration tasks	- use FNscript commands to provide the proxy with the authentication information to submit to the service provider - configure your application to tell proxy what type of authentication is required - if you are using FusionXML authentication, make sure your partner application knows how to interpret the proxy's authentication document

The remainder of this chapter describes this use case in more detail and provides procedures for performing the tasks required for authentication.

Authenticating a Proxy with a Service Provider

When a proxy sends a service provider application a request to perform an activity, the application may require authentication from the proxy before accepting the request and performing the activity.

Authentication message flow

The following flow of documents and messages takes place when a proxy sends a request to a service provider that requires authentication:

1. The proxy sends an application document to the service requestor.
2. The service requestor rejects the document, and returns a message with:

- a "401 Unauthorized" error in the message body
- A WWW-Authenticate header message specifying the type of authentication (Basic or FusionXML) that the service provider requires from the proxy.

3. The proxy resubmits the application document with the appropriate application user information included in the HTTP message header.

   The specific information the proxy supplies depends on the type of authentication that the application specifies:

   - For Basic authentication, the proxy returns a string containing a user name and password.
   - For FusionXML authentication, the proxy returns a string containing an XML authentication document containing additional user information.

   You must ensure that the proxy is configured to know the information to submit. For information about configuring a proxy with application authentication information, see "Providing User Information to the Proxy" on page 100.

4. The service provider validates the proxy's user information; if the information is accepted, the service provider:

   - creates a session for the proxy
   - performs the activity
   - returns an application document to the proxy indicating completion

5. The proxy generates a command document to notify the Fusion engine that the activity was completed.

From the point at which the service provider accepts the proxy's application document (Step 4), the procedure for performing the activity and sending the required notifications is the same as any for any service provider application.

For information about sending synchronous requests to a services provider, see Chapter 3, "Synchronous Service Provider Use Case." For information about sending asynchronous requests, see Chapter 4, "Asynchronous Service Provider Use Case."

Enabling authentication

To enable the service provider to authenticate the proxy and complete the activity, take the following steps:

1. Configure your partner application to require authentication and to specify the type of configuration required.
2. Configure the proxy with the authentication information for the service provider.

The following sections describe how to perform these tasks.

For more information about proxy configuration and session authentication, see the *Forte Fusion Backbone System Guide*.

Configuring the Service Provider for Authentication

To require that a proxy authenticate itself with a service provider application, the application must be configured to inform the proxy what kind of authentication is required. The specific procedures you need to follow depend, of course, on the type of your application.

You specify the authentication type in the WWW-Authenticate header of the 401 Unauthorized error message that the application sends in response to the original application document. To accomplish this task, include either of the following strings in the message header (where *backbonename* and *proxyname* represent the Fusion Backbone and service provider's partner proxy):

Basic authentication

```
WWW-Authenticate: basic realm="backbonename:proxyname"
```

Basic authentication instructs the proxy to send the application a user name and password. The proxy sends the name and password of the user you specify when you configure the proxy, as described in "Providing User Information to the Proxy" on page 100.

The user name and password are encoded with Base64 and embedded as an authorization header in the HTTP message containing the application document requesting the application to perform the activity. For information about the format of this message header, see "Sending a User Name and Password" on page 93.

FusionXML authentication

```
WWW-Authenticate: FusionXML realm="backbonename:proxyname"
```

FusionXML authentication instructs the proxy to send the application an XML authentication document with additional information, for example, a user profile or role. The authentication document is encoded with Base64 and embedded as a string in the authorization header of the HTTP message containing the application document that requests the application to perform the activity.

The proxy constructs the authentication document based on the application user information you supply when you configure the proxy, as described in the following section, "Providing User Information to the Proxy."

Note

Your application must be capable of interpreting the authentication document. For information on the authentication document XML structure and vocabulary, see Appendix B, "Proxy Document Element Hierarchies" and the Fusion Backbone online Help.

Providing User Information to the Proxy

For either Basic or FusionXML authentication, the proxy must know what application user information to pass to the service provider when requested. The proxy can only send the information about which it knows, so be sure that you configure values for all items that the service provider expects.

You use FNscript commands to provide application user information to the proxy. The following table shows the relevant commands:

Command	Description
SetAplSession username [password [, otherinfo]]	Specifies the name of a user authorized to connect to the service provider application; may contain an optional password and “otherinfo” string with information meaningful to the application, for example, a department or manager name.
AddAplRole rolename	Specifies the name of a user role that the application requires for access. For example, a CreditCheck application might only allow access to proxies that connect with the role of CreditManager.
SetAplProfile profilename	Specifies the name of a user proflle required by the application for access.

For more information about proxy configuration, see the *Forte Fusion Backbone System Guide*.

Chapter 7

Proxy Recovery Use Case

This chapter describes the XSL stylesheets and related integration tasks required to submit activities to a service provider application after its proxy has failed and recovered.

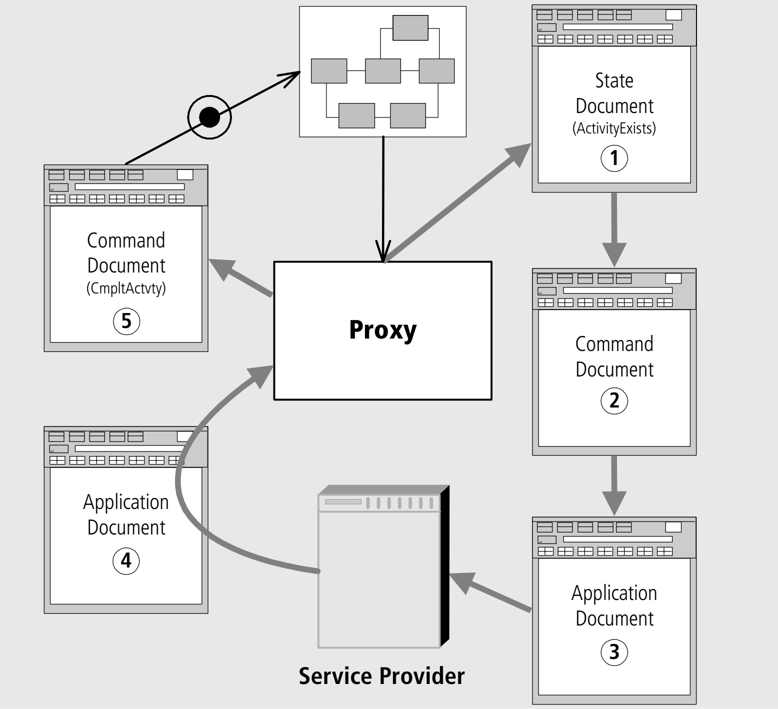
The chapter also presents an alternative scenario in which the recovered activities are aborted.

For information about the basic tasks and stylesheets required for a proxy to submit a request to a service provider, see Chapter 3, "Synchronous Service Provider Use Case." If the proxy and application communicate asynchronously, also see Chapter 4, "Asynchronous Service Provider Use Case."

If you are running an independent proxy, you cannot use the recovery methods that Fusion provides. A proxy must be connected to a running Fusion process engine to be able to recover activities after proxy or application failure.

Use Case Summary

The following table provides an overview of this use case:

Use Case Information	Description
description	A service provider proxy fails then recovers during a Fusion process.
expected outcome	Upon recovery, the proxy retrieves all ACTIVE activities from the engine and submits them to the partner application as though they were new activities.
actors	▮ application proxy ▮ Fusion process engine ▮ service provider application
proxy document flow	**1** After it recovers, the proxy generates an "ActivityExists" state document for each activity in the ACTIVE state. **2** Proxy generates command document with request to service provider for each activity. **3** Proxy sends application document to service provider to perform activity. **4** Service provider sends application document to proxy with disposition of activity. **5** Proxy generates command document to notify engine to complete activity.  State Document (ActivityExists) 1 Command Document (CmpltActvty) 5 Proxy Command Document 2 Application Document 4 Service Provider Application Document 3
required stylesheets	outbound stylesheet to: ▮ transform state document into command document to send request to service provider (Step 2) ▮ generate application document containing request to service provider (Step 3) inbound stylesheet to send command document to engine to complete the activity (Step 5)
other integrator tasks	▮ provide your partner application with the logic to handle possibly duplicate activities

The remainder of this chapter describes this use case in more detail and provides procedures for writing the stylesheets required for its successful completion.

Note For an alternative scenario in which the activity is aborted rather than submitted to the application, see "Alternative Processing: Aborting the Recovered Activity" on page 108.

Submitting a Recovered Activity to a Service Provider

If a proxy for a service provider fails during a Fusion process, the following actions occur:

1. The backbone manager restarts the proxy.
2. The proxy reconnects to the Fusion process engine.
3. The proxy retrieves a list of ACTIVE activities from the engine.
4. The proxy generates a state document for each activity with a value of “ActivityExists” for the State attribute of the FNCndState element.

The proxy, however, has no way of knowing whether or not any of these activities were sent to the service provider before the failure. Therefore, your outbound stylesheet must contain rules for processing these recovered activities and handling them according to the partner application’s business logic.

The simplest way to handle a recovered activity is to send the service provider an application document with a request to perform the activity. In essence, you are treating the activity like any other activity that has entered the ACTIVE state, without regard for whether the activity might already have been sent to the service provider.

The option is generally preferred for situations where it is acceptable for the application to perform the activity twice, or where the application can determine how to handle duplicate requests. For example, if the activity involved adding records to a database, the database itself would reject any duplicate records.

Submitting the Activity

To send an ACTIVE activity to a service provider after a proxy recovers, your outbound stylesheet must:

- locate the activity in the state document by matching an FNCndState element whose State attribute has a value of “ActivityExists”
- generate a command document that sends an application document requesting the service provider to perform the activity

To accomplish these tasks, include a template like the following in your outbound stylesheet:

```
Match found activity in state doc   <xsl:template match="FNCndState[@State=’ActivityExists’]">
Send message                          <FNAplCommand Command="SendMessage" Method="Post"/>
Specifies that app doc follows          <FNMessage>
Document element for app doc              <WorkRoot>
Top-level element for app doc               <NewWork>
Process children                              <xsl:apply-templates/>
                                            </NewWork>
                                          </WorkRoot>
                                        </FNMessage>
                                      </FNAplCommand>
                                    </xsl:template>
```

The procedure for submitting the activity to the service provider is similar to that for handling a newly ACTIVE activity. The only difference is that for the recovered activity your stylesheet must match an FNCndState State attribute value of “ActivityExists,” as shown in the above template, rather than “ActivityStarted,” as for new activities.

Synchronous and asynchronous activities

Because you are handling the recovered activity like a new activity, you should treat the recovered activity as synchronous or asynchronous according to the usual processing for this proxy and application. That is, apply your usual rules for including or omitting information from the FNIdentity element to identify the activity.

For the complete procedure required to generate the command document and application document to a synchronous service provider, see "Communicating Synchronously With a Service Provider" on page 57.

For the complete procedure required to generate the command document and application document to an asynchronous service provider, see "Communicating Asynchronously With a Service Provider" on page 70.

Generated Document Examples

The following sections describe the XML documents that are generated when a service provider proxy recovers from failure.

State Document

A proxy that has recovered from failure generates a state document like the following for each activity that the Fusion process engine informs it is in the ACTIVE state:

Note The only difference between a state document for an activity in the ActivityExists state and a state document for an activity in the ActivityStarted state is the value of the State attribute of the FNCndState element.

XML declaration

```
<?xml version="1.0"?>
```

State doc document element

```
<FNState>
```

State of activity

```
  <FNCndState State="ActivityExists">
```

Process/activity information

```
    <FNIdentity
        ProcessID="52"
          ProcessName="FNOrdersProcess"
          ActivityID="5"
          ActivityName="BillPreparation"
          ActivityAppCode="Billing"/>
```

Process attribute list from app dict.

```
    <FNProcessAttributeList>
```

Status process attribute

```
      <FNProcessAttribute Name="Status" Type="TextData">
```

Value from previous activity

```
        CreditVerified
      </FNProcessAttribute>
```

StartingMessage proc attribute

```
      <FNProcessAttribute Name="StartingMessage"
                             Type="TextData">
```

Service requestor app document

```
        <MsgDoc>
          <NewOrder>
            <Atts>
              <Att>
                 <AttName>Billee</AttName>
                 <AttType>TextData</AttType>
                 <AttValue><![CDATA[fred]]></AttValue>
              </Att>
              <Att>
                 <AttName>Shippee</AttName>
                 <AttType>TextData</AttType>
                <AttValue><![CDATA[fred]]></AttValue>
              </Att>
              <Att>
                 <AttName>ItemCount</AttName>
                 <AttType>TextData</AttType>
                 <AttValue><![CDATA[20]]></AttValue>
              </Att>
              <Att>
                 <AttName>OrderID</AttName>
                 <AttType>TextData</AttType>
                <AttValue><![CDATA[15]]></AttValue>
            </Att>
          </Atts>
          </NewOrder>
        </MsgDoc>
      </FNProcessAttribute>
    </FNProcessAttributeList>
  </FNCndState>
</FNState>
```

Document Notes

In the above document, the value of the Status process attribute is CreditVerified, because that was the last activity that updated the Status value.

Command Document

The command document that the proxy generates to send a recovery activity to a service provider is identical to the document generated for a newly ACTIVE activity. For an example of such a command document, see “Command Document” on page 77.

Application Document

The application document in this case also is identical to the application document generated for a newly ACTIVE activity. For a discussion of this application document, see “Application Document” on page 79.

Receiving Notification of Completion from the Application

When the application notifies the proxy that it has performed the activity, the proxy uses the inbound stylesheet to generate a command document as it would for any other completed activity. Whether or not the proxy treats the activity as synchronous or asynchronous depends on how the activity was submitted to the service provider.

For the steps required to generate the command document based on an asynchronous response from the application, see “Receiving Synchronous Notification of Completion” on page 64.

For the steps required to generate the command document based on an asynchronous response from the application, see “Receiving Asynchronous Notification of Completion” on page 80.

Alternative Processing: Aborting the Recovered Activity

If a process is interrupted because a proxy fails, you might need to abort or rollback the activity, or even abort the process, without submitting the activity to the application. Such a situation might exist, for example, in a manufacturing process in which timing is critical.

In such a case, your outbound stylesheet needs a rule to generate a command document to instruct the engine to abort the activity or process, rather than generating an application document.

To accomplish this task, your outbound stylesheet needs rules to:

1. Generate the FNCommand document element for command documents.
2. Find an activity in the state document whose State attribute has a value of “ActivityExists.”
3. Generate a command to the engine to abort the process or activity.
4. If the proxy processes activities asynchronously, include the identity of the activity, as specified by the FNIdentity element in the state document.

To accomplish these tasks, include a template like the following in your outbound stylesheet:

Match found activity in state doc
Doc element for command doc
Send command to engine
Copy FNIdentity from state doc

```
<xsl:template match="FNCndState[@State=’ActivityExists’]">
  <FNCommand>
    <FNCndCommand Command="AbortActivity"/>
      <xsl:copy-of select="FNIdentity"/>
    </FNCndCommand>
  </FNCommand>
</xsl:template>
```

Transformation Notes

In the above template:

- To roll back the activity or abort the entire process, substitute “RollbackActivity” or “AbortProcess,” respectively, for the value of the Command attribute.
- If the proxy processes activities synchronously, omit the FNIdentity information.

Altering the Document Flow

This processing option implements an alternative flow of proxy documents. In this case, the documents flow from the Fusion process engine to the proxy, then back to the engine, without involving the partner application.

Also, although templates to generate commands to the engine usually are placed in inbound stylesheets, the above template is in the proxy’s outbound stylesheet. The reason for this difference is that the source document is an outbound state document, rather than an inbound application document.

Generated Document Examples

The following sections provide examples of the XML documents that a proxy generates for the preceding scenario.

State Document

For an example of a state document that a proxy generates for a recovered activity, see “State Document” on page 105.

Command Document

The following command document is generated by the preceding steps.

Start command document
Command to send msg to app
Copy FNIdentity from state doc

```
<FNCommand>
  <FNCndCommand Command="AbortActivity">
      <FNIdentity
        ProcessID="52"
          ProcessName="Replication"
          ActivityID="5"
          ActivityName="Cloning"
          ActivityAppCode="CloneDNA"/>
  </FNCndCommand>
</FNCommand>
```

Document Notes

In the above document, the value of the Command attribute depends on the template you specify in your outbound stylesheet.

Application Document

There is no application document generated for this variation of the use case. The proxy transforms the state document into a command document that sends a command back to the Fusion process engine, rather than to the application.

Chapter 8

Application Recovery Use Case

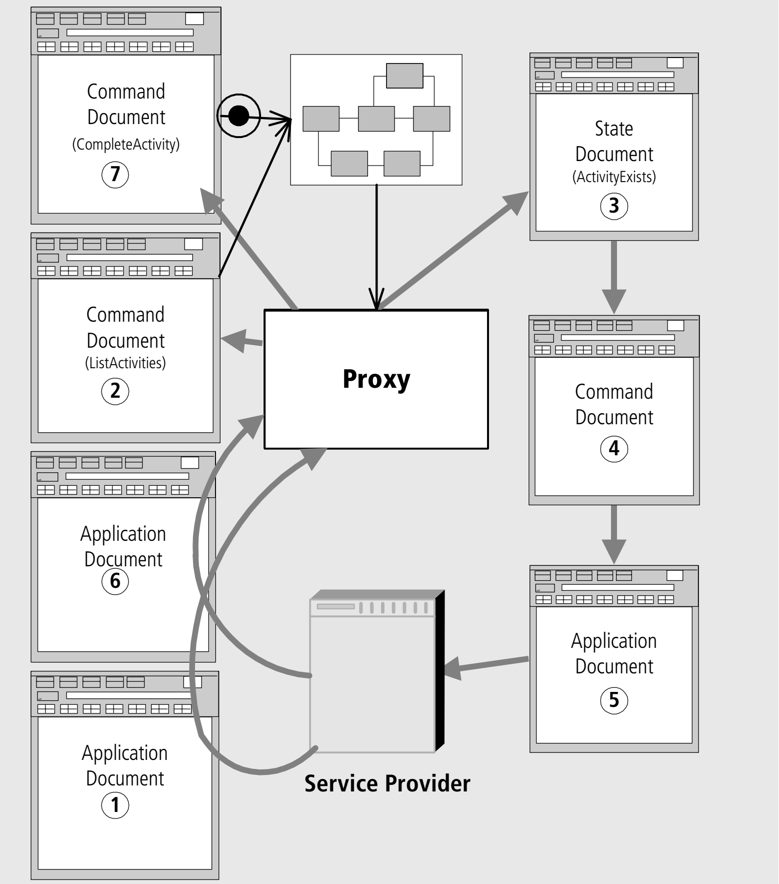
This chapter describes the XSL stylesheets you need to write and related integration tasks you need to perform to submit activities to a service provider application that has failed and recovered.

For information about the basic tasks and stylesheets required for a proxy to submit a request to a service provider, see Chapter 3, “Synchronous Service Provider Use Case.” If the proxy and application communicate asynchronously, also see Chapter 4, “Asynchronous Service Provider Use Case.”

If you are running an independent proxy, you cannot use the recovery methods that Fusion provides. A proxy must be connected to a running Fusion process engine to be able to recover activities after proxy or application failure.

Use Case Summary

The following table provides an overview of this use case:

Use Case Information	Description
description	A service provider application fails during and recovers during a Fusion process.
expected outcome	When the application notifies the proxy that it has recovered, the proxy retrieves all ACTIVE activities and submits them to the partner application.
actors	▮ service requestor application ▮ application proxy ▮ Fusion process engine
proxy document flow	**1** Upon recovery, service provider sends application document to notify proxy that it is available again. **2** Proxy generates a "ListActivities" command document to retrieve all ACTIVE activities. **3** Proxy generates an "ActivityExists" state document for each activity in the ACTIVE state. **4** Proxy generates command document with request to service provider for each activity. **5** Proxy sends application document with request for each activity to service provider. **6** Service provider sends application document to proxy with disposition of activity. **7** Proxy generates command document to notify engine that activity was performed.  Command Document (CompleteActivity) 7 State Document (ActivityExists) 3 Command Document (ListActivities) 2 Proxy Command Document 4 Application Document 6 Application Document 5 Service Provider Application Document 1

Use Case Information	Description
required stylesheets	inbound stylesheet to: - transform recovery notification from application into request to list ACTIVE activities (Step 2) - transform application document into command document to notify engine that application performed the activity (Step 7) outbound stylesheet to: - transform state document into command document that sends request to service provider (Step 4) - generate application document containing request to service provider (Step 5)
other integrator tasks	Configure the proxy for the service provider as an HTTP server. For details about configuring proxies, see the *Forte Fusion Backbone System Guide*. Provide your partner application with the logic to: - notify the proxy that the application has recovered - handle possibly duplicate activities resubmitted by the proxy

The remainder of this chapter describes this use case in more detail and provides procedures for writing the stylesheets required for its successful completion.

Notifying the Proxy of Application Recovery

A service provider application that has failed must notify its proxy when it becomes available again, so that it can continue to receive requests to perform work. The form of this notification depends on the application; it might be as simple as an application document with a single element, for example:

```
<MsgDoc>
  <AppRecovered>
</MsgDoc>
```

The proxy's inbound stylesheet should include two templates. The first template:

- matches the application document root element
- creates the FNCndCommand element in the command document
- causes all children of the root element in the application document to be processed

The second template:

- matches the application's recovery notification
- transforms this notification into a command document to list all currently ACTIVE activities for the proxy

The stylesheet also should include a template that:

To accomplish these tasks, include the first template shown below, and a template like the second, in your inbound stylesheet:

Match app doc root element

Create FNCndCommand element

Process all elements in app doc

Match recovery notification in app doc

Generate ListActivities command

```
<xsl:template match="/">
  <FNCommand>
    <FNCndCommand>
      <xsl:apply-templates/>
    </FNCndCommand>
  </xsl:template>

<xsl:template match="AppRecovered">
  <FNCndCommand Command="ListActivities"/>
</xsl:template>
```

Generated Command Document Example

The following command document is generated by the preceding template.

Start command document

Command to send msg to app

```
<FNCommand>
  <FNCndCommand Command="ListActivities"/>
</FNCommand>
```

Submitting Activities to a Recovered Application

When the proxy receives the command document with the “ListActivities” command, it checks its list of ACTIVE activities. For each activity on the list, it generates a state document with a State attribute value of “ActivityExists.”

Your outbound stylesheet now can generate command documents from these state documents to submit these activities to the recovered service provider application, just as it would if the proxy had failed and recovered.

For information about submitting recovered activities to a service provider, see “Submitting a Recovered Activity to a Service Provider” on page 103.

Receiving Notification of Completion from the Application

When the application notifies the proxy that it has performed the activity, the proxy uses the inbound stylesheet to generate a command document as it would for any other completed activity. Whether or not the proxy treats the activity as synchronous or asynchronous depends on how the activity was submitted to the service provider.

For the steps required to generate the command document based on an asynchronous response from the application, see “Receiving Synchronous Notification of Completion” on page 64.

For the steps required to generate the command document based on an asynchronous response from the application, see “Receiving Asynchronous Notification of Completion” on page 80.

Chapter 9

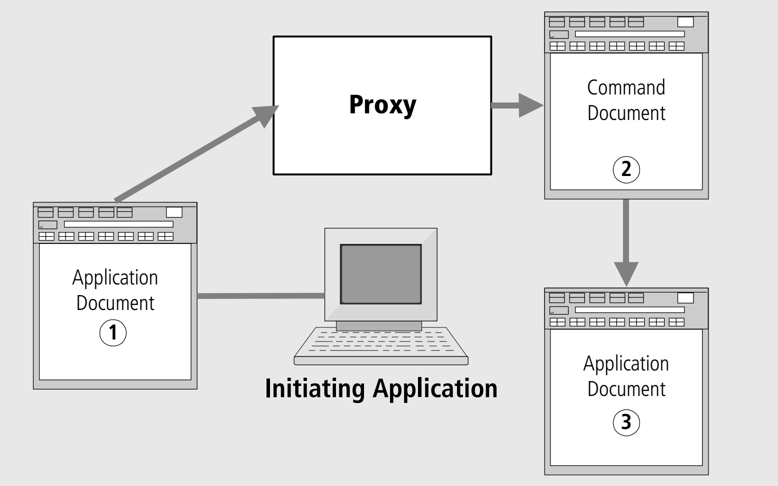
Independent Proxy Use Case

This chapter describes the XSL stylesheets you need to write any related integration tasks you need to perform to enable an application to send information in the form of XML documents to another application through an independent Fusion proxy. An independent proxy is one that is not connected to a Fusion process engine.

If the proxy requires user authentication from the requestor application, there are additional tasks you must perform. After you read this chapter, see Chapter 10, “Independent Proxy Authentication Use Case.”

Use Case Summary

The following table provides an overview of this use case:

Use Case Information	Description
description	An application sends XML data to another application through a Fusion proxy that is not connected to a running Fusion process engine.
expected outcome	The proxy receives an application document from the initiating application and uses the inbound stylesheet to transform the document into another application document, which it transmits to a target application.
actors	▮ initiating application ▮ application proxy ▮ target application
proxy document flow	**1** Initiating application sends application document to proxy. **2** Proxy generates command document with message for target application. **3** Proxy sends message as application document to target application.  Proxy Command Document 2 Application Document 1 Initiating Application Application Document 3
required stylesheets	inbound stylesheet to transform application document into command document that generates message containing application document for target application No outbound stylesheet is required because, without a Fusion process engine, there is no state document generated.
other integration tasks	Set the **UseProcessEngine** FNscript command to **off** to specify that the proxy is not connected to a Fusion process engine.

The remainder of this chapter describes this use case in more detail and provides procedures for writing the stylesheet required for its successful completion.

Transferring Data Between Applications

You can use a Fusion proxy to enable applications to share data by transforming XML documents. For such simple data transformations, the proxy—known as an *independent proxy*—need not be connected to a Fusion process engine.

Independent proxies always involve client-server applications, that is, a client application performs some work, then sends data to a server.

For example, you might have an online registration page for visitors to a trade show. When a new visitor registers, you might want to send some of the information to an application that prints a badge for the visitor. You would not need to create a Fusion process definition to perform such a simple transfer of data.

To transfer data between applications through an independent proxy:

- Create an inbound stylesheet to transform the initiating application's application document into a command document containing an application document for the target application.
- Configure the proxy to specify that it is not connected to a Fusion engine.

The rest of this chapter describes how to create such a stylesheet and configure an independent proxy.

For general information about using independent proxies, see "Communicating Between Applications Without A Process Definition" on page 40.

Authentication Note

The following steps assume that any user is authorized to connect to the proxy and the applications involved. For information about how to allow only authorized users to create connections, see Chapter 10, "Independent Proxy Authentication Use Case."

Creating a Stylesheet for Data Transformation

To transfer data between applications through an independent proxy, create an XSL stylesheet that performs the following functions:

- includes a template that overrides the default action, which is to copy all text nodes to the results document
- creates the FNCommand document element to identify itself as a command document
- creates the FNAplCommand element to send a message to the target application
- creates the FNDestination element to specify the location of the target application
- creates the FNMessage element that encompasses the application document to be sent to the target application
- creates a document element for the target application document
- transforms any relevant data from the initiating application document into the appropriate format within the application document for the target application

The following sections describe how to create a stylesheet that performs these functions.

Because you are transforming the inbound application document, an independent proxy uses only an inbound stylesheet to generate the application document for the target application. There is no outbound stylesheet because, in the absence of a Fusion process engine, there is no state document to transform into a command document to the application.

Step 1. Overriding the Default for Text Nodes

By default, the Fusion XSL processor concatenates the values of all text nodes and copies them to the results document. This behavior occurs even if you do not explicitly include the default template that causes it. When transforming data between applications, however, this behavior is generally undesirable. Rather, you want to specify which text nodes from the initiating application document are copied to the target application document.

For example, the printer application may have no need of personal visitor information, such as a home address, that is part of the visitor record in the registration application. You only want to transmit information about the visitor that the printer application needs to print a badge.

▶ **To copy text nodes selectively to the results document:**

1 Include templates in your stylesheet to match the specific text you want to appear in the results document.

  For an example of such templates, see "Specifying the Message Content" on page 123.

2 In your inbound stylesheet, include the following template in place of the usual default for text nodes:

```
<xsl:template match="text()"/>
```

  This "empty" template matches any text node. However, rather than performing some action on the match, as the default template does, it simply ends (with the "/"), doing nothing.

Step 2. Generating the Command Document Element

The proxy identifies a command document by using the FNCommand element as its document element. The first task your inbound stylesheet must perform is to generate this document element.

To generate the FNCommand document element, include the following template in your inbound stylesheet:

Match document element
Create command doc doc elem.
Process children of root element

```
<xsl:template match="RegDoc">
  <FNCommand>
    <xsl:apply-templates/>
  </FNCommand>
</xsl:template>
```

Transformation Notes

In the above template:

- The xsl:apply-templates element ensures that the XSL processor processes all children of the document element of the application document. Without the xsl:apply-templates element, the XSL processor would complete its work after it created the FNCommand element, generating it as an empty element.

Step 3. Generating the Command to Send a Message

The next step your outbound stylesheet must perform is to generate the commands to send a message to the application.

▶ **To generate these commands:**

- Create a template that matches the Registrant document element in the initiating application document.
- Generate an FNAplCommand element whose Command attribute has a value of “SendMessage” and whose Method attribute specifies the HTTP method (Get or Post).

For more information about specifying values for the Command and Method attributes of the FNAplCommand element, see the Fusion Backbone online Help.

To generate the appropriate FNAplCommand element and attributes, include a template like the following in your outbound stylesheet:

Match a new employee record
Send an HTTP message

```
<xsl:template match="Registrant">
  <FNAplCommand command="SendMessage" method="Post">
    <xsl:apply-templates/>
  </FNAplCommand>
</xsl:template>
```

Transformation Notes

The above template includes an xsl:apply-templates element to ensure that the following information, to be specified later, is placed within the FNAplCommand element:

- optionally, the target application location information (generated by the template in the next section, “Specifying the Target Application Location”)
- the FNMessage element that holds the target application document (generated by the template in “Generating a Message to the Target Application” on page 122)
- the target application document itself (generated by the templates in “Specifying the Message Content” on page 123)

Communicating with Multiple Applications

You can specify that an application send messages to multiple target applications. To do so, include one FNAplCommand element for each target application; each FNAplCommand must contain the appropriate location information and content transformations, as described in the following sections.

Step 4. Specifying the Target Application Location

The next task the stylesheet can perform is to specify the location to which to send the message. You accomplish this task by including an FNDestination element within the FNAplCommand element.

To specify the location of the target application, include a template like the following in your inbound stylesheet:

Created in Step 3.
Specify the target app URL
Process Registrant children

```
<xsl:template match="Registrant">
  <FNAplCommand command="SendMessage" method="Post">
    <FNDestination address="badger.acmecorp.com:4500"/>
    <xsl:apply-templates/>
  </FNAplCommand>
```

Step 5. Generating a Message to the Target Application

The next step is to generate an FNMessage element to hold the target application document. The FNMessage element is a child of the FNAplCommand element, located at the same level in the hierarchy as the FNDestination element created in Step 4.

To create the FNMessage element, include the following template in your inbound stylesheet:

Created in Step 3.
Created in Step 4.
Start of msg to target app
Msg content (see Step 7.)

```
<xsl:template match="Registrant">
  <FNAplCommand command="SendMessage" method="Post">
    <FNDestination address="badger.acmecorp.com:4500"/>
    <FNMessage>
      ...application document for target application
    </FNMessage>
  </FNAplCommand>
```

Step 6. Generating the Target Application Document Element

The next step is to generate the document element that informs the defect tracking application that it is receiving a new user record. You accomplish this task by generating a NewUser element within the template that matches the Registrant element—which is the document element of the initiating application document—as shown in the previous section.

To generate the document element for the target application document, including a template like the following in your inbound stylesheet:

```
<xsl:template match="Registrant">
  <FNAplCommand command="SendMessage" method="Post">
    <FNDestination address="badger.acmecorp.com:4500"/>
    <FNMessage>
      <NewBadge>
        <xsl:apply-templates/>
      </NewBadge>
    </FNMessage>
  </FNAplCommand>
```

Created in Step 3.
Created in Step 4.
Created in Step 5.
Target app document element
ProcessRegistrant children
(see Step 7.)

Step 7. Specifying the Message Content

After you create the FNMessage element, you create its contents, which become the application document sent to the target application.

The application document contains whatever information the application expects. In the example used in this chapter, assume that the target application—the badge printing application—requires the following information to set up a new user:

- visitor name
- visitor number
- company name
- job title

The XSL stylesheet must contain the appropriate transformations to render this information in a format that is usable by the target application.

To generate the appropriate data, include templates like the following in the inbound stylesheet:

Match RegistrantName child of Registrant
Create BadgeName element
Combine first, last names

```
<xsl:template match="RegName">

  <BadgeName>
    <xsl:value-of select=’concat(First," ",Last)’/>
  </BadgeName>
```

Match RegNumber child of Registrant
Create BadgeNumber element
Copy value of RegNumber

```
<xsl:template match="RegNumber">
  <BadgeNumber>
    <xsl:value-of select="."/>
  </BadgeNumber>
</xsl:template>
```

Match Company child of Registrant
Create Employer element
Copy value of Company element

```
<xsl:template match="Company">
  <Employer>
    <xsl:value-of select="."/>
</xsl:template>
```

Match JobTitle child of Registrant
Create Position element
Copy value ofRegistrant

```
<xsl:template match="JobTitle">
  <Position>
    <xsl:value-of select="."/>
  </Position>
</xsl:template>
```

Transformation Notes

In the above templates:

- The xsl:apply-templates element in the template that matches Registrant (shown in “Generating the Target Application Document Element” on page 123) ensures that all its children elements are processed; the results of any matches are placed within the NewBadge document element in the target application document.
- The value of the BadgeName element is generated by matching the RegistrantName element in the source document, then uses the “concat” function to create a string consisting of the First child element, a blank, and the Last child element.

  Concatenation is an example of the many XPath functions that are available for manipulating string and numerical data. For detailed information about XPath functions, see the XPath specification at *http://www.w3.org/TR/xpath.html*.
- To create the BadgeNumber element, the stylesheet matches the VisitorNumber element in the source document. The “.” in the xsl-value-of transformation specifies that the value of the BadgeNumber element should be the value of the current element, which is VisitorNumber.

  The same logic applies to creating Employer from Company and Position from JobTitle.

Sample Documents and Stylesheet

The following sections provide examples of:

- an application document from an initiating application
- the inbound stylesheet to transform this XML document
- the command document the proxy generates, including the application document sent to the target application

Initiating Application Document

The following XML document provides an example of a new employee record that a personnel application might create.

xml declaration

Document document

```
?xml version="1.0"?>
<RegDoc>
  <Registrant>
    <RegName>
      <First>Frank</First>
      <Middle>X.</Middle>
      <Last>Jones</Last>
    </RegName>
    <RegNumber>10203045</RegNumber>
    <HomeAddress>
      <HStreet>6666 South Milagro Boulevard</HStreet>
      <HCity>San Arnaldo</HCity>
      <HState>WY</HState>
      <HZip>74949-2423</HZip>
    </RegAddress>
    <Company>Superior Products</Company>
    <WorkAddress>
      <WStreet>1000 Superior Way</WStreet>
      <WCity>Superior</WCity>
      <WState>WY</WState>
      <WZip>74801-2003</WZip>
    </WorkAddress>
    <JobTitle>Customer Support Representative</JobTitle>
  </Registrant>
</RegDoc>
```

Inbound Stylesheet

The following XSL stylesheet provides the transformations described in this chapter to transmit application documents between an initiating application and a target application.

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                  version="1.0"/>
<xsl:output method="xml" indent="yes"/>

<xsl:template match=”text()”/>
```

```
<xsl:template match=”/”>
  <xsl:apply-templates/>
<xsl:template>

<xsl:template match=”RegDoc”>
  <FNCommand>
    <xsl:apply-templates/>
  </FNCommand>
</xsl:template>

<xsl:template match="Registrant">
  <FNAplCommand command="SendMessage" method="Post">
    <FNDestination address="bugs.acmecorp.com:4500"/>
    <FNMessage>
      <NewBadge>
        <xsl:apply-templates/>
      </NewBadge>
    </FNMessage>
  </FNAplCommand>
</xsl:template>

<xsl:template match="RegName">
  <BadgeName>
    <xsl:value-of select=’concat(First," ",Last)’/>
  </BadgeName>
</xsl:template>

<xsl:template match="RegNumber">
  <BadgeNumber>
    <xsl:value-of select="."/>
  </BadgeNumber>
</xsl:template>

<xsl:template match="Company">
  <Employer>
    <xsl:value-of select="."/>
  </Employer>
</xsl:template>

<xsl:template match="JobTitle">
  <Position>
    <xsl:value-of select="."/>
  </Position>
</xsl:template>

</xsl:stylesheet>
```

Command Document

The proxy generates the following command document based on the XSL stylesheet shown in this chapter.

```
                               <?xml version="1.0"?>
Start command document         <FNCommand>
Command to send msg to app       <FNAplCommand Command="SendMessage" Method="Post">
Location of target app             <FNDestination address="badger.acmecorp.com:4500"/>
Application document follows       <FNMessage>
Document element of app doc          <NewBadge>
                                       <BadgeName>Frank Jones</BadgeName>
                                       <BadgeNumber>10203045</BadgeNumber>
                                       <Employer>Superior Products</Employer>
                                       <Position>Customer Support Representative</Position>
                                     </NewBadge>
Application doc completed          </FNMessage>
                                 </FNAplCommand>
                               </FNCommand>
```

Target Application Document

The lines between the start and end tags of the FNMessage element in the above command document make up the application document that is sent to the target application. The application document begins and ends with the start and end tags, respectively, of the NewUser document element.

Configuring an Independent Proxy

To configure a proxy to function as an independent proxy, issue the following FNscript command:

```
UseProcessEngine off
```

Issue this command in addition to any normal configuration commands.

Note Independent proxy always refers to the proxy whose partner application is the initiating application, not the target application.

You can issue this command even if the proxy has been configured previously to use a Fusion process engine (with the **SetCEngine** command). In this way, you can override the use of the engine, and use the proxy for test purposes.

For more information about configuring an independent proxy, see the *Forte Fusion Backbone System Guide*.

Chapter 10

Independent Proxy Authentication Use Case

This chapter describes the integration tasks required to enable an independent proxy—a proxy that is not connected to a running process engine—to require user authentication from its partner application. It also describes how to require authentication from the proxy by a target application. Authentication discussed in this chapter applies only to proxies configured for HTTP communication.

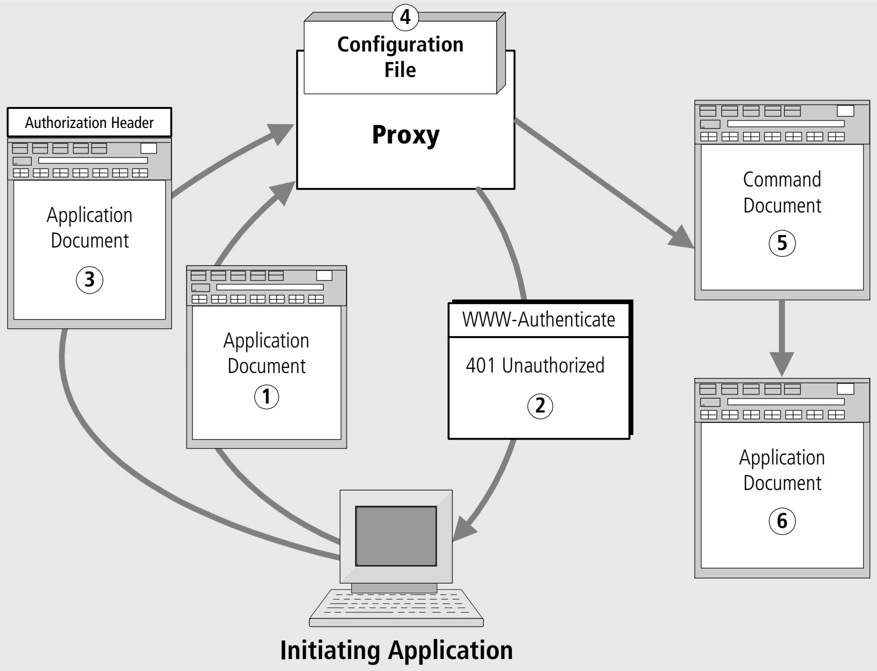
For information about the basic tasks and stylesheets required when using an independent proxy, see Chapter 9, “Independent Proxy Use Case.”

For general information about using independent proxies, see “Communicating Between Applications Without A Process Definition” on page 40.

For general information about authentication in a Fusion enterprise application (including authentication for proxies configured for JMS), see the *Forte Fusion Backbone System Guide*.

Use Case Summary

The following table provides an overview of this use case:

Use Case Information	Description
description	An independent proxy requires authentication from its partner application.
expected outcome	The proxy accepts the initiating application’s authentication and transforms its application document into a message for the target application.
actors involved	- the application proxy - the initiating application
proxy document and message flow	**1** Initiating application attempts to send XML application document to its proxy. **2** Proxy responds with message containing HTTP 401 Unauthorized error and WWW-Authenticate header requesting Basic authentication. **3** Application resends application document with authorization header containing user name/password combination. **4** Proxy verifies authentication information against data in configuration file. **5** Proxy accepts initiating application document and uses inbound stylesheet to generate command document to send message to target application **6** Proxy sends application document to target application.
	 4 Configuration File Proxy Authorization Header Application Document 3 Application Document 1 WWW-Authenticate 401 Unauthorized 2 Command Document 5 Application Document 6 Initiating Application
stylesheets required	inbound stylesheet to: - transform initiating application document into command document that sends message to target application (Step 5) - transform relevant data in initiating application document into application document usable by target application(Step 6)
Other integration tasks	- use **UseProcessEngine** FNscript command specify that proxy is independent - use **SetAuthentication** FNscript command to specify authentication type - use **SetCredentials** FNscript command to specify user name and password against which the proxy validates authentication information sent by initiating application

The remainder of this chapter describes this use case in more detail and provides procedures for performing the tasks required for authentication.

Authenticating an Application To an Independent Proxy

You can use a Fusion proxy—known as an independent proxy—to transmit XML data between two applications without being connected to a process engine. As part of this process, you might want to require the initiating application to authenticate itself with the proxy, so that only authorized users can submit data to the target application.

Note Alternatively, the target application can request authentication from the proxy before accepting the data from the initiating application. For information about such a configuration, see “Authenticating a Proxy To a Target Application” on page 134.

Authentication Message Flow

The following flow of documents and messages takes place when an initiating application sends a request to a proxy that requires authentication:

1. The initiating application sends an application document to the proxy.
2. The proxy rejects the document, and returns a message with:
   - a “401 Unauthorized” error in the message body
   - A WWW-Authenticate header message specifying that the proxy requires Basic authentication
3. The application resubmits the application document with the user name and password encoded in the HTTP Authentication message header.

Note You could submit the user information with the application document from the beginning and bypass Step 2 and Step 3. However, it is good practice for security reasons not to send the authorization information until the proxy requests it. An HTTP client always be prepared for a 401 response.

4. The proxy validates the application’s user information against the values in its configuration file; if the information is accepted, the proxy:
   - creates an HTTP session for the application
   - accepts the application document from the initiating application
   - sends a cookie to the application, so that the application is authorized for subsequent requests during the current HTTP session

   You must ensure that the proxy is configured with the information against which to validate this user name and password. For information about configuring a proxy with application authentication information, see “Configuring the Proxy for Authentication” on page 133.

From this point, the procedure for transmitting the XML to the target application is the same as described in Chapter 9, “Independent Proxy Use Case.”

Enabling Authentication

To enable the initiating application to be authenticated and the application document accepted, take the following steps:

1. Enable the application to send the authentication information when requested by the proxy.
2. Configure the proxy to require Basic authentication, and provide the proxy with the authentication information to use for validation.

The following sections describe how to perform these tasks.

Submitting Authentication Information

When a proxy that requires authentication returns a 401 Unauthorized message, you must resubmit the initiating application document with the appropriate authorization information in the message header. Independent proxies only support Basic authentication, for which you include a *user name:password* string in the authorization header of the application document as follows:

```
Authorization: basic base64(username:password)
```

where *username* and *password* correspond to the values with which you configured the proxy with the **SetCredentials** FNscript command, as described in "Specify the Authentication Values" on page 133.

For example, assume that user name "personnel" and password "$$entry##" are a valid user name and password known to the proxy. When the proxy prompts the initiating application to submit authentication, you would include the following string in the HTTP message header:

```
Authorization: basic cGVyc29ubmVsOiQkzw50cnkjIw==
```

The name and password are encoded with Base64 and sent to the proxy, which validates them against the information in the proxy's configuration file. If the proxy validates this user name and password, it processes the initiating application document.

Configuring the Proxy for Authentication

For an independent proxy to authenticate an initiating application document, the proxy must:

- be configured as independent
- require Basic authentication on the local server
- know the authentication values to validate

The following subsections describe the FNscript commands you use to accomplish these tasks. For more information about any of these commands, and for additional information about configuring proxies, see the *Forte Fusion Backbone System Guide*.

Step 1. Specify the Proxy As Independent

To specify that a proxy not require connection to a running process engine, issue the following FNscript command:

```
UseProcessEngine off
```

Even if a proxy has been configured with a process engine (with the **SetCEngine** command), you can use the above command to specify that the proxy does not need to be connected to that engine. This situation is useful, for example, if you simply want to test the proxy without running the full Fusion application.

Step 2. Specify That the Proxy Require Authentication

To specify that the proxy require authentication from its partner application, issue the following FNscript command:

```
SetAuthentication Scheme=Basic Server=Local
```

Basic authentication specifies that only a user name and password are required for authentication. Local authentication specifies that the authentication information is stored in the proxy's configuration file on the local server, rather than in a UserValidation associated with a engine.

Step 3. Specify the Authentication Values

To specify the values against which the proxy should validate the user name and password that the application supplies, issue the following FNscript command:

```
SetCredentials user=username password=password
```

where *username* and *password* represent the authentication values expected from the application. These values are stored in encrypted form in the proxy's configuration file and validated at run time.

Authenticating a Proxy To a Target Application

Rather than having the initiating application authenticate itself to the independent proxy, you might configure your Fusion application so that the proxy must authenticate itself to the target application before the target application accepts any data. In such a case, you configure the proxy in the same manner as that described in "Configuring the Proxy for Authentication" on page 133.

However, you must configure your target application to request authentication when the proxy attempts to transmit the target application document. To accomplish this task, code your target application to send a 401 Unauthorized message to the proxy, along with the following WWW-Authenticate message header:

```
WWW-Authenticate: basic realm="backbonename:proxyname"
```

where *backbonename* is the name of the Fusion Backbone and *proxyname* is the name of the independent proxy. When the proxy receives the request for authentication, it resends the target application document with the appropriate authorization header as described in "Submitting Authentication Information" on page 132, supplying the user name and password that was specified with the **SetAplSession** FNscript command.

Note You could configure your Fusion application so that both the independent proxy and the target application require authentication. However, they would both need to accept the same user name and password, because you can only configure the proxy with one set of authorization information.

Appendix A

Transforming Proxy Documents

This appendix illustrates how the Fusion proxy:

- transforms inbound and outbound proxy documents according to its XSL stylesheets
- can function as an HTTP server or client within the Fusion enterprise application
- can function as a JMS listener or sender within the Fusion enterprise application

To understand how XSL transformations occur, see Chapter 1, “Introduction.” For procedures describing how to develop the stylesheets required, see the chapters in this manual for the relevant use cases.

For an overview of proxy documents, including management of HTTP sessions and JMS sessions, see the *Forte Fusion Backbone System Guide*.

Proxy Document Processing

All XML documents contain patterns consisting of elements, attributes, text data, and instructions. Elements identify the type of content and can be followed by attributes (name-value pairs). Fusion allows its applications to send XML documents without restrictions on form or content.

The proxy has an internal XSL processor to manage transformation of incoming and outgoing documents.

XSL Stylesheets for the Proxy

Fusion application proxies refer to their configured XSL stylesheets to translate between the XML understood by the application (or adapter) and that understood by the proxy. For example, the result of applying an XSL stylesheet to an application document is a command document. Command documents are internal to the proxy and drive the proxy's interaction with its partners (the Fusion process engine and the application).

Through transformation, XML documents arriving from the application (application documents) result in command documents that determine operations performed by the process engine. Events arriving from the engine are converted into state documents and result in command documents that determine the XML sent to the application.

Figure 5 illustrates a proxy taking an application document and inbound XSL stylesheets as input, and generating a command document as the output:

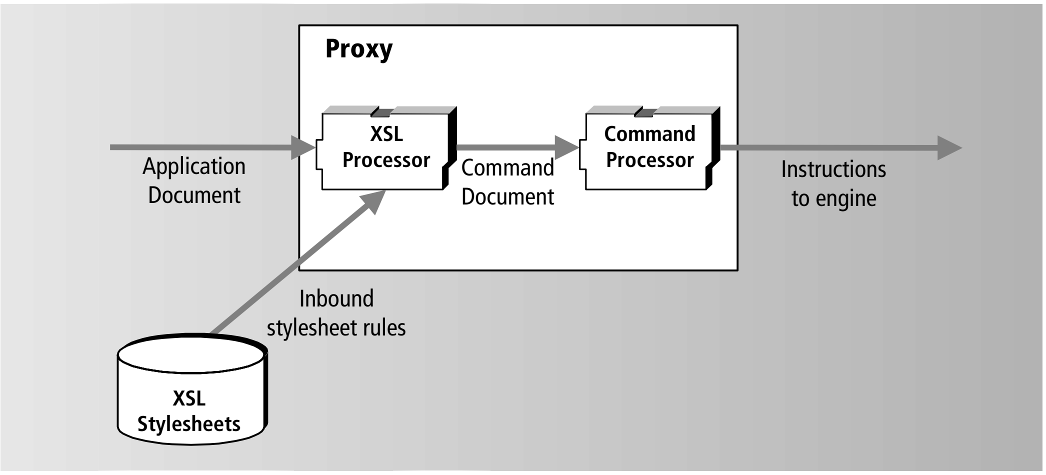


Figure 5 *Generating Command Documents*

A proxy configured for HTTP sessions can present itself to its application as an HTTP server, HTTP client, or both. When configured for JMS sessions, the proxy can present itself as a JMS listener, JMS sender , or both.

The following sections explain typical message processing according to the proxy's function in the Fusion system. The scenario illustrates the process definition in the Fusion TOOL adapter example.

For an overview of proxy document types and concepts, see the *Forte Fusion Backbone System Guide*.

Service Requestor Application

Service requestor applications send requests to the proxy. The proxy can act as an HTTP server, processing service requestor HTTP requests and sending back responses. It can also act as a JMS listener processing service requests. The processing of proxy documents results in the specified actions being performed by the Fusion process engine and the application, typically initiating a business process or starting/completing a business activity.

Figure 6 shows an order processing flow in which a Web client initiates a new order by forwarding an XML representation of the order to its proxy. The proxy's XSL processor uses its inbound stylesheets to identify certain patterns in the order and determine which business process instance to create. The proxy also sets initial values for Fusion process attributes, based on the XML representation of the order.

Figure 6 illustrates the order processing flow described subsequently in numbered steps. Step numbers are indicated by circled numbers on the diagram:

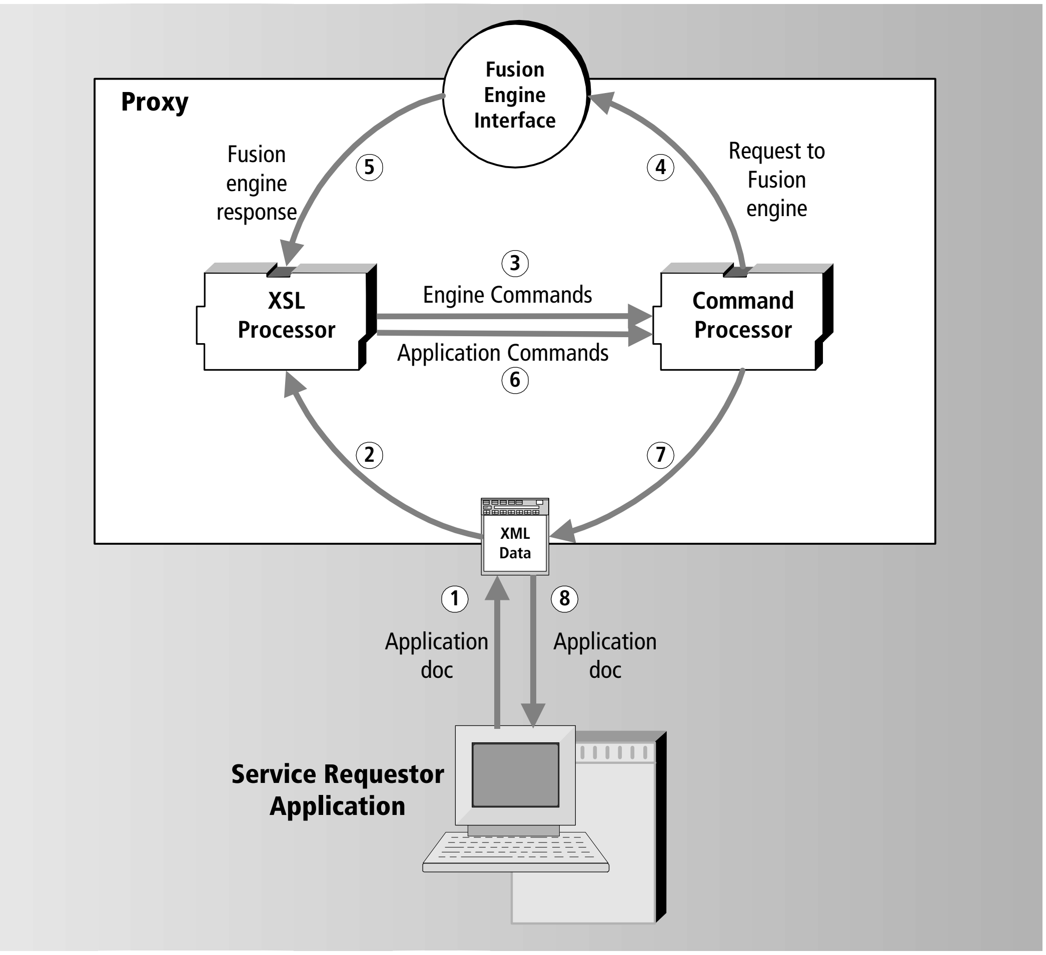


Figure 6 Fusion Order Entry Service Requestor Sends in New Order

Note This is an example of one possible message flow; many variations are possible in an actual system.

1 The service requestor application sends a request to the proxy. The request contains an application document to place a new order:

```
<PlaceOrder>
  <ShipTo>
    Beelzebub
  </ShipTo>
</PlaceOrder>
```

2 The proxy forwards the document to its XSL processor.

3 The XSL processor applies the inbound stylesheets:

XML declaration
XSL stylesheet declaration

Locate source element

Command to
Create process
Process children

Send response to application (HTTP sessions)

```
<?xml version="1.0">
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">
  <xsl:template match="PlaceOrder"/>
    <FNComment>Create a new process</FNComment>
    <FNCommand>
      <FNCndCommand Command="CreateProcess"
                    ProcessName="OrderEntry">
      <xsl:apply-templates/>
      </FNCndCommand>
      <FNAplCommand Action="SendResponse"/>

    </FNCommand>
  </xsl:template>
</xsl:stylesheet>
```

The XSL processor then generates a command document to start the process instance and send a response to the application:

command doc document elem.
Command to create
the OrderEntry process
Create list of process attributes

Send response to application (HTTP sessions)

```
<FNCommand>
  <FNCndCommand Command="CreateProcess"
              ProcessName="OrderEntry">
      <FNProcessAttributeList>
        <FNProcessAttribute Name="Shipto"
              Type="TextData">
              Beelzebub
        </FNProcessAttribute>
      </FNProcessAttributeList>
  </FNCndCommand>
  <FNAplCommand Command="SendResponse"/>

</FNCommand>
```

4 The command processor interprets the command document and invokes the CreateProcess command.

5 The process engine returns the CreateProcess status, which indicates that it has started the process instance. The proxy represents this information in a state document:

State of the process

Identity of the process

```
<FNCndState State="ProcessStarted">
  <FNIdentity ProcessName="OrderEntry"
              ProcessID="1234"/>
</FNCndState>
```

6 The XSL processor applies the outbound stylesheet to translate this output for the response message:

HTTP sessions use SendResponse

JMS sessions use SendMessage

Process the children

```
<xsl:template match=
              "/FNState/FNCndState[@State=’ProcessStarted’]">
  <FNAplCommand Command="SendResponse">
    <FNMessage>
      <OrderEntered>
        <xsl:apply-templates/>
      </OrderEntered>
    </FNMessage>
  </FNAplCommand>
</xsl:template>
<xsl:template match=
              ""FNCndState[@State=’ProcessStarted’]/FNIdentity">
  <Cfnumber>
  <xsl:value-of select="@ProcessID"/>
  </Cfnumber>
</xsl:template>
```

7 The command processor continues processing the inbound application document. It finds the <FNAplCommand> and sends an outbound response document regarding the process status.

8 The application receives the XML confirmation message:

Proc. ID as confirmation nmbr

```
<Order>...
  <Conf>
    1234
  </Conf>
</Order>
```

After the Fusion process engine starts the process, the first activity in this example is a credit check. The ensuing steps are described in the following section.

Service Provider Application

When a proxy acts as an HTTP client or JMS listener, it creates application documents that are sent to the application's URL, and takes action based on the response. The first document is sent when the Fusion process engine informs the proxy that it can start a business activity. The proxy starts the activity and sends a message describing the request to the application.

In the credit check application example, the proxy's session parameters are configured to identify it to the engine as the credit check application (the subscriber to work events involving credit checking). The process engine presents the proxy with each request for a customer credit check. The proxy builds an XML state document based on this information, forwarding each message to the XSL processor. The XSL processor generates an application command document to be sent to the application.

In the original command document, the tags specify process attributes, such as the customer number and the amount. The service provider's response document includes a tag that indicates either approval or rejection. The XSL processor handles this response, which instructs the process engine interface to pass the activity completion status (approved or not) to the engine by invoking the CompleteActivity method.

Figure 7 shows how the CreditCheck proxy might process a request for a credit check.

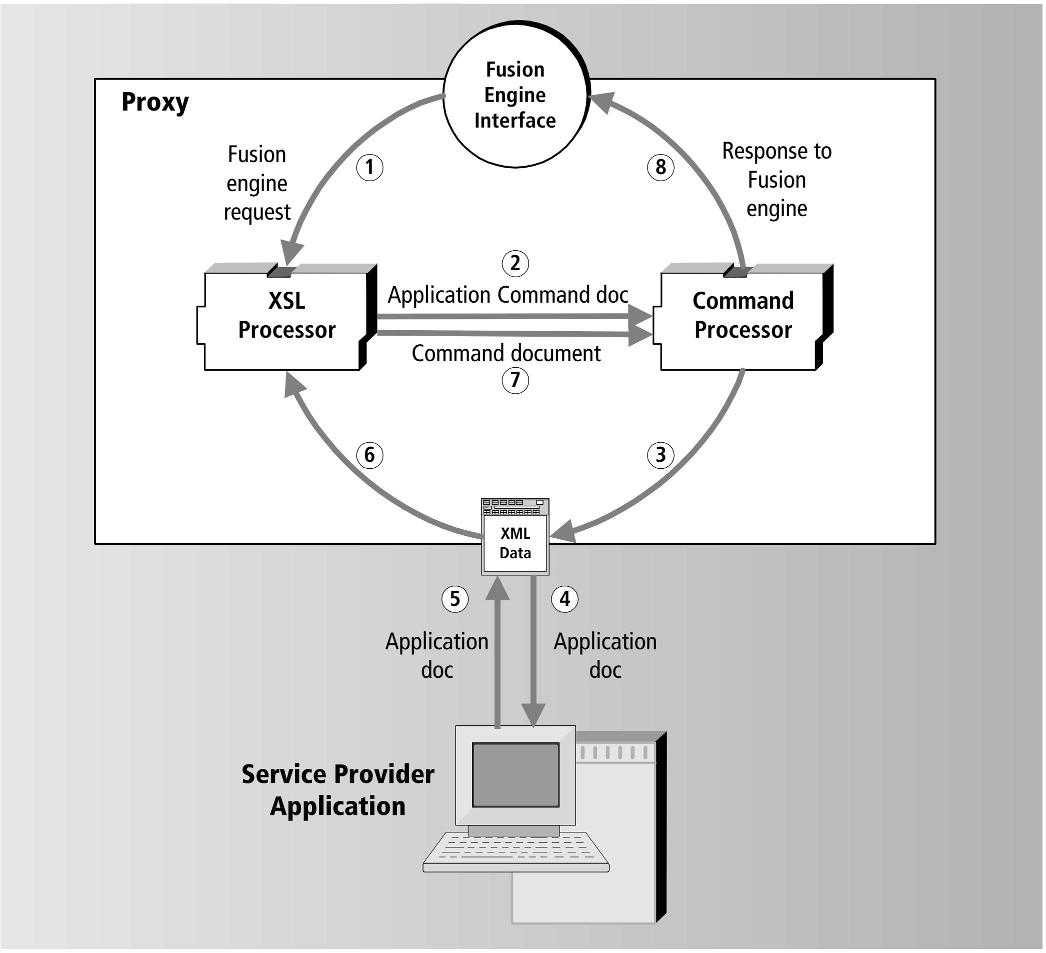


Figure 7 *Fusion StartAndVerify Application Processes Credit Check*

1 The process engine starts the activity in response to the READY event. The proxy then generates an ActivityStarted state document, which provides the parameters needed to perform the credit check:

State of the activity

Identity of the activity

List of process attributes

Credit denial

```
<FNState>
  <FNCndState State="ActivityStarted">
    <FNIdentity ProcessID="920"
              ActivityID="2"
              ActivityName="CreditCheck"
              ActivityAppCode="Perform Credit Check"/>
    <FNProcessAttributeList>
      <FNProcessAttribute Name="Billee"
              Type="TextData">
              B_1
      </FNProcessAttribute>
      <FNProcessAttribute Name="ItemCount"
              Type="TextData">
              100
      </FNProcessAttribute>
      <FNProcessAttribute Name="CreditApproved"
              Type="TextData">
              No
      </FNProcessAttribute>
    </FNProcessAttributeList>
  </FNCndState>
</FNState>
```

2 The XSL processor checks its outbound XSL stylesheet and applies the rules that match. In this example, the first relevant rule specifies that when an activity with a value of ActivityStarted is received, the proxy should notify the application to process the new work:

Matches a started activity

Send message to application

Start of the message

Tell app to process new work

Process child elements

```
<xsl:template match="FNCndState/[@State=’ActivityStarted’]"/>
<FNAplCommand Command="SendMessage"
              Method="Post">
  <FNMessage>
    <WorkRoot>
      <NewWork>
        <xsl:apply-templates/>
      </NewWork>
    </WorkRoot>
  </FNMessage>
</FNAplCommand>
</xsl:template>
```

The XSL processor applies the rules and generates a command document with the application command. The process ID and activity ID are required in this scenario only if the interaction between the proxy and the application is asynchronous.

Document element

Command to send message to application

Start of the message

Identifies the started activity

Fusion process attributes transformed into application's vocabulary

```
<FNCommand>
  <FNAplCommand Command="SendMessage"
               Method="Post">
    <FNMessage>
      <WorkRoot>
        <NewWork>
          <ProcessID>920</ProcessID>
          <ActivityID>2</ActivityID>
          <WorkType>CreditCheck</WorkType>
          <Atts>
            <Att>
              <AttName>Billee</AttName>
              <AttType>TextData</AttType>
              <AttValue>B_1</AttValue>
            </Att>
            <Att>
              <AttName>ItemCount</AttName>
              <AttType>TextData</AttType>
              <AttValue>100</AttValue>
            </Att>
            <Att>
              <AttName>CreditApproved</AttName>
              <AttType>TextData</AttType>
              <AttValue>No</AttValue>
            </Att>
          </Atts>
        </NewWork>
      </WorkRoot>
    </FNMessage>
  </FNAplCommand>
</FNCommand>
```

3. The command processor builds an outbound application document from the command document.
4. The application document transmits the request and the application performs the credit check.
5. The application responds with a message to disapprove the credit.

6. The incoming application document is forwarded to the XSL processor:

```
<WorkRoot>
  <WorkCompleted>
    <ProcessID>920</ProcessID>
    <ActivityID>2</ActivityID>
    <Atts>
      <Att>
        <AttName>Billee</AttName>
        <AttType>TextData</AttType>
        <AttValue>B_1</AttValue>
      </Att>
      <Att>
        <AttName>ItemCount</AttName>
        <AttType>TextData</AttType>
        <AttValue>100</AttValue>
      </Att>
      <Att>
        <AttName>CreditApproved</AttName>
        <AttType>TextData</AttType>
        <AttValue>Deadbeat</AttValue>
      </Att>
    </Atts>
  </WorkCompleted>
</WorkRoot>
```

Customer tagged as deadbeat

7 The XSL processor applies its inbound stylesheet, and generates a CompleteActivity command document:

Document element

Engine command

Identify the activity

Fusion process attribute list

```
<FNCommand>
  <FNCndCommand Command="CompleteActivity">
    <FNIdentity ProcessID="920"
             ActivityID="3"/>
    <FNProcessAttributeList>
      <FNProcessAttribute Name="Billee"
             <Type="TextData">
             B_1
      </FNProcessAttribute>
      <FNProcessAttribute Name="ItemCount"
             Type="TextData">
             100
      </FNProcessAttribute>
      <FNProcessAttribute Name="CreditApproved"
             Type="TextData">
             Deadbeat
      </FNProcessAttribute>
    </FNProcessAttributeList>
  </FNCndCommand>
</FNCommand>
```

8 The proxy engine interface interprets the command document and performs the specified CompleteActivity API request to the process engine.

Appendix B

Proxy Document Element Hierarchies

This appendix provides diagrams showing the XML hierarchies for command documents, state documents, and authentication documents, including element attributes.

Command Document Element Hierarchy

Figure 8 illustrates the hierarchy of elements that the proxy uses to construct command documents:

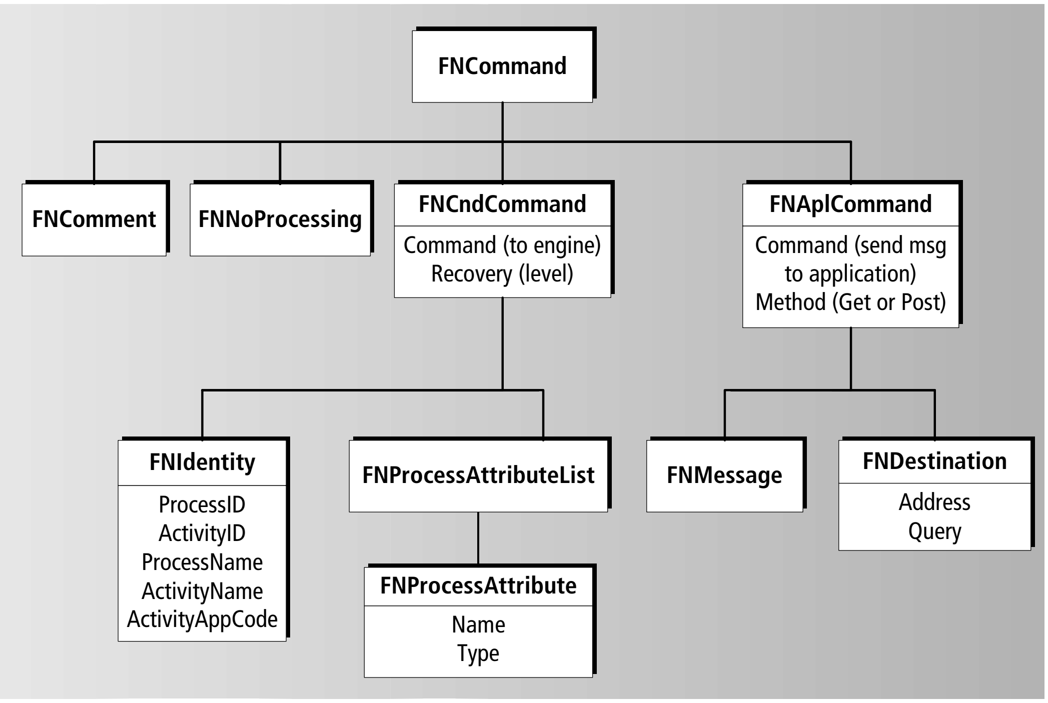


Figure 8 *Command Document Hierarchy*

You need to be familiar with the command document hierarchy so that:

- your inbound stylesheets can transform application documents into command documents
- your outbound stylesheets can transform state documents into command documents

For detailed information about the command document elements and attributes, see the Fusion Backbone online help.

State Document Hierarchy

Figure 9 illustrates the hierarchy of elements that the proxy uses to construct state documents:

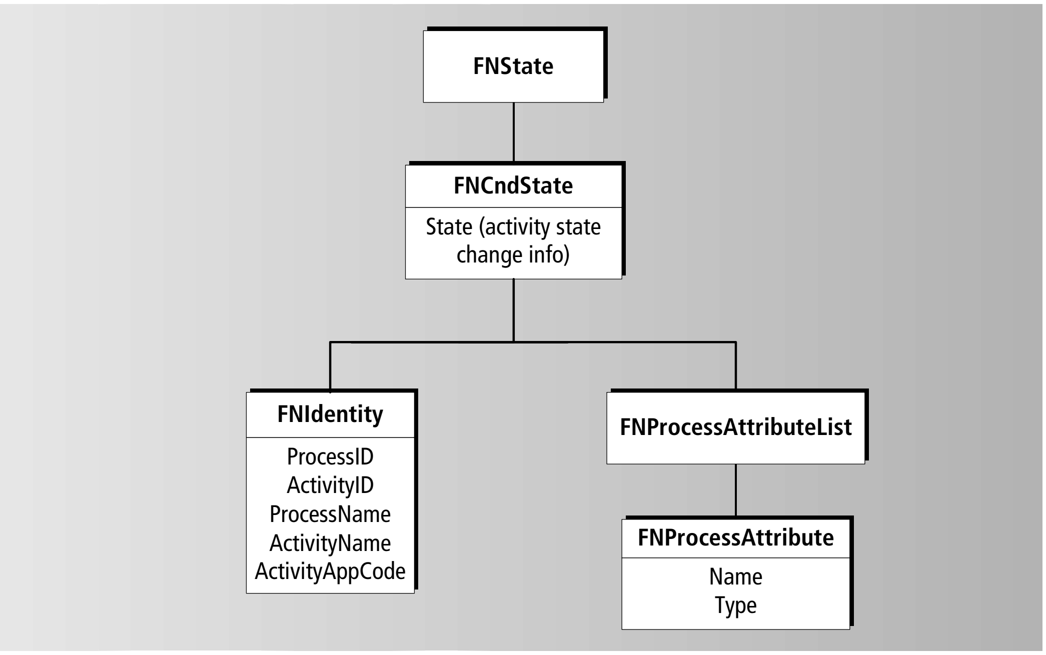


Figure 9 *State Document Element Hierarchy*

You need to be familiar with the state document hierarchy so that your outbound stylesheets can transform state documents into command documents.

For detailed information about the state document elements and attributes, see the Fusion Backbone online help.

Authentication Document Hierarchy

Figure 10 illustrates the hierarchy of elements used to construct authentication documents:

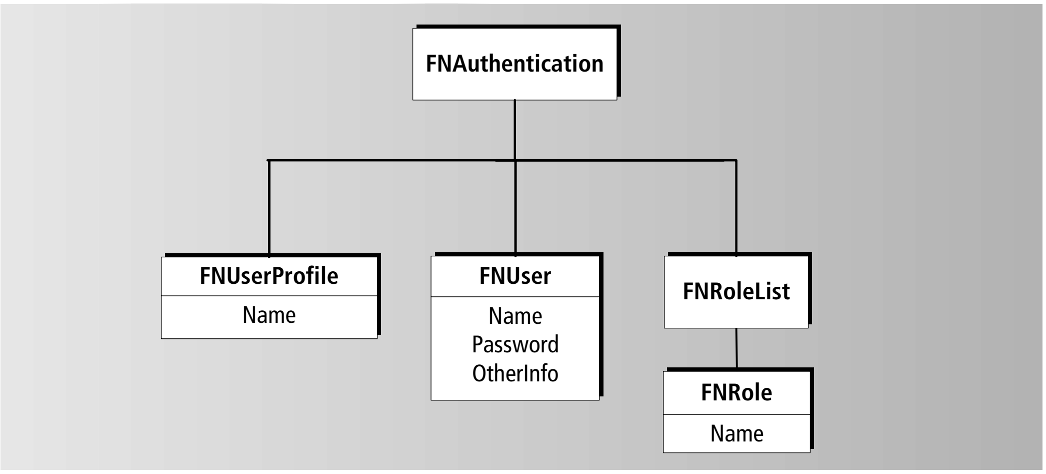


Figure 10 *Authentication Document Element Hierarchy*

Authentication documents apply only to proxies configured to communicate using HTTP. Authentication documents (and authentication in general) are optional, and all child elements within the documents are optional. Any values you provide are placed in the appropriate fields of the user profile used by the proxy.

Proxies create authentication documents when an application asks the proxy to authenticate itself. You create authentication documents when an application must authenticate itself to a proxy.

You need to be familiar with the authentication document hierarchy so that you can create authentication documents when required by a proxy. You do not need to write stylesheets to transform authentication documents, because you construct them using the above vocabulary and structure, which the proxy already understands.

For more information on authentication documents, see the *Forte Fusion Backbone System Guide*. For detailed information about authentication document elements and attributes, see the Fusion Backbone online help.

Index