

ClearCommerce Software
Using the XML Input Component 5.9



Copyrights

This manual contains proprietary information that is protected by copyright. The information in this manual is subject to change without notice. No part of this manual may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, for any purpose other than the licensee's personal use without prior written permission of eFunds Retail Solutions. The software described in this manual is furnished under a license granted by eFunds Retail Solutions to the licensee. This software may be used or copied only in accordance with the terms of the license agreement.

(c) 2000-2006 by eFunds Retail Solutions. **All rights reserved.**

The ClearCommerce software uses Expat-XML Parser Toolkit, which is subject only to the Mozilla Public License Version 1.1 (the "License"), and the source code is available only under the terms of the License; you may not use the Expat-XML Parser Toolkit except in compliance with the License. Software distributed under the License is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License at <http://www.mozilla.org/MPL/> for the specific language governing rights and limitations under the License. The Original Code is Expat-XML Parser Toolkit, which is available at <http://www.jclark.com/xml/expat.html> The Initial Developer of the Original Code is James Clark. Portions created by James Clark are © 1998, 1999 James Clark. All Rights Reserved.

The Microsoft Windows version of the ClearCommerce software uses the Pthreads-Win32 - POSIX 1003 Threads Library for Win32, © 1998, subject to the GNU Library General Public License, version 2. Source code for this library and the GNU Library General Public License are included on the CD. This library is free software; you can redistribute it and/or modify it only under the terms of the GNU Library General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version. This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Library General Public License for more details. You should have received a copy of the GNU Library General Public License along with this library; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA. This library provides an implementation of PThreads based upon the POSIX 1003.1c-1995 (POSIX.1c) standard. Contributors are listed in the file "MAINTAINERS" located at: <ftp://sources.redhat.com/pub/pthreads-win32/sources/pthreads-snap-2000-09-08/>

The ClearCommerce software uses the IBM ICU 2.4 Program Copyright © 1995-2002, International Business Machines Corporation and others. All Rights Reserved. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the abovecopyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

The ClearCommerce software contains BSAFE which is licensed to eFunds Retail Solutions by RSA Data Security, Inc. © 2003 RSA Data Security, Inc. All rights reserved.

Portions of this software are copyrighted by DataDirect Technologies, 1991-2003

The ClearCommerce software contains SSL Plus which is licensed to eFunds Retail Solutions by Consensus Development Corporation. © 1997-2000 Consensus Development Corporation. All rights reserved. Portions are © 1997-1998 Consensus Development Corporation, a wholly owned subsidiary of Certicom Corp. All rights reserved. Contains an implementation of NR signatures, licensed under U.S. patent 5,600,725. Protected by U.S. patents 5,787,028; 4,745,568; 5,761,305. Patents pending.

The ClearCommerce software uses Flex © 1990 The Regents of the University of California. All rights reserved. This code is derived from software contributed to Berkeley by Vern Paxson. Flex includes software developed by the University of California, Berkeley and its contributors. The United States Government has rights in this work pursuant to contract no. DE-AC03-76SF00098 between the United States Department of Energy and the University of California.

Trademarks

ClearCommerce® and FraudShield® are registered trademarks of eFunds Retail Solutions. PaymentDirector™ is a trademark of eFunds Retail Solutions.

Adobe® and Acrobat® are registered trademarks and PostScript™ is a trademark of Adobe Systems Incorporated.

BSAFE® is a registered trademark of RSA Data Security, Inc.

Quova, the Quova logo and GeoPoint are service marks of Quova, Inc.

SSL Plus™ is a trademark of Consensus Development Corporation.

ZIPsales™ is a trademark of DPC Computers, Inc.

Other marks cited in this document are the property of their respective owners.

Support

eFunds Retail Solutions is committed to the ongoing support of its products as documented in the license agreement.

If you need features or functionality that are not currently offered by eFunds Retail Solutions, or if you have questions about the product or need assistance, contact Technical Support by phone at 414-341-3207 or by e-mail at ccsupport@efunds.com or at the ClearCommerce Software Customer Support Web site <http://www.clearcommerce.com/support/>

Notices

Fraud Protection: Fraud protection systems, such as those offered by eFunds Retail Solutions, can help a fraud case manager better focus investigative efforts by identifying which transactions exhibit traits similar to those that have been correlated with incidences of fraud in the past. No fraud system can definitively determine whether any given transaction is, in fact, fraudulent. Therefore, fraud protection systems can form only one part of a comprehensive business decision-making process that involves human oversight and investigation of each transaction in question. The responsibility to instill such a review process lies solely with each individual merchant and commerce service provider and not with eFunds Retail Solutions.

Documentation

The documentation for this product is also available in softcopy format in the installation package.

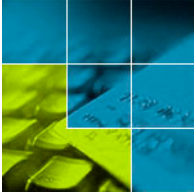
The softcopy documentation is provided in .pdf format, and may be viewed or printed using Adobe Acrobat Reader. If you do not have Adobe Acrobat Reader installed on your system, you can download the correct version for your platform from the following Web site:

<http://www.adobe.com/prodindex/acrobat/readstep.html>

Install Acrobat Reader 6.0 or later for best viewing results and print to a PostScript Level 2 (or higher) printer for best printing results.

Refer to the Release Notes for late additions, corrections, and revisions to the documentation.

Pubrev 110306

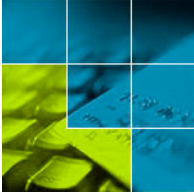


Contents

About This Manual	ix
Audience	ix
Contents	ix
Highlighting and Notes	x
Product Documentation	x
Online Resources	xiii
Chapter 1. Introduction	1
What is XML?	1
Java and XML.....	1
Simple API for XML (SAX)	2
Document Object Model (DOM).....	3
Java API for XML Parsing (JAXP)	3
HTTP POST	4
Java Secure Socket Extension (JSSE)	4
XML and the ClearCommerce Engine	4
Validating Your XML Using the OrderFormDoc DTD	5
XML Web Sites and Tools	5
XML Web Sites	5
Java and XML Tools.....	6
Chapter 2. XML Interface	7
XML Input to the ClearCommerce Engine.....	7
XML and the ClearCommerce API Document Hierarchy.....	8
Sample XML Document.....	11
PreAuth XML Example	13

Chapter 3. Sample Client Applications.....	15
Sample HTML Client	16
Sample Java Clients	20
Read XML document from a file and send to ClearCommerce Engine	20
run.bat File Example	20
make.bat File Example	21
XMLtoEngineFromFile.java Example	22
Create an XML document with JAXP and send to ClearCommerce Engine	26
Sample input.txt File.....	28
Sample XMLtoEngine.java File	29
 Chapter 4. XML Input Component Security	 51
Security Protocols	51
Trusted Server Certificate	52
How To Get and Install a Secure Server Certificate.....	52
Using a Temporary Trusted Certificate	54
Configuring the Java Client for a Temporary Certificate	55
Configuring a Web Browser for a Temporary Certificate	56
Security Certificate Web Sites	56
 Chapter 5. XML Input Component Configuration	 57
Configuring the XML Input Component Table	57
Configuring the XML Input Component Command Line Keys	59
 Chapter 6. XML Input Component Functionality.....	 65
White Space Handling	65
XML Processing Instructions	65
XML Encoding Declaration.....	66
 Chapter 7. Troubleshooting	 67
Confidence Checks	67
Errors	69
Corrective Actions	70

Index.....73



About This Manual

This manual describes the XML Input Component and how to use it to send data from a Web site (or storefront) to the ClearCommerce Engine.

Audience

This manual is intended for system integrators and other developers who will program the interface between the storefront and the ClearCommerce Engine.

Contents

This manual includes the following information:

Chapter 1, “Introduction,” which provides an overview to XML and how the XML input component is used with the ClearCommerce Engine.

Chapter 2, “XML Interface,” which describes how to send XML data to the Engine.

Chapter 3, “Sample Client Applications,” which contains sample code and HTML for implementing XML.

Chapter 4, “XML Input Component Security,” which discusses security protocols and trusted server certificates.

Chapter 5, “XML Input Component Configuration,” describes how to configure the database and the Director so that the component can be enabled.

Chapter 6, “XML Input Component Functionality,” which highlights some of the functionality associated with the XML input component that the client application needs to be aware of.

Chapter 7, “Troubleshooting” which provides information about debugging the XML input component.

Highlighting and Notes

The following highlighting styles are used in this manual:

- **Bold** indicates commands, command-line options, and interface controls, such as the names of icons, menus, menu items, buttons, check boxes, and tabs.
- *Italic* indicates variables that you must replace with a value. It also indicates book titles and emphasized terms.
- `Monospace` indicates file names and code examples.

The following note styles are used in this manual:

Note: Information that further explains a concept or instruction in the main text.

Important: Information that you might overlook within the main text and that is essential for the completion of a task.

Caution: Information that either alerts you or tells you how to avoid a potential loss of data or a system failure.

Product Documentation

The documentation in the ClearCommerce software product library includes printed manuals, softcopy manuals, and HTML help systems. The softcopy manuals (in .pdf format) are packaged with the products, either in the installation image or on a separate CD. The HTML Help systems can be accessed from each page of the browser-based Store Administrator Tool, System Administrator Tool, and Account Administrator (CSP) Tool.

The product library includes the documents that are described in the following table.

Documentation	Description
<i>Store Administrator Guide</i>	General reference information about ClearCommerce Engine. How-to information about setting up, configuring, and managing an online store that is hosted using ClearCommerce Engine.
Online Help for the Store Administrator Tool	How-to information about performing configuration and management tasks using the Store Administrator Tool. To access Help, click the Help link from any page of the Store Administrator Tool.
<i>Risk Manager Guide</i>	Reference and how-to information about using eFunds' ClearCommerce® FraudShield® to protect against fraudulent orders.
<i>System Administrator Guide</i>	How-to information about performing configuration, management, and administration tasks.
Online Help for the System Administrator Tool	How-to information about performing configuration, management, and administration tasks using the System Administrator Tool. To access Help, click the Help link from any page of the System Administrator Tool.
<i>Installation Guide</i>	How-to information about installing ClearCommerce software, including pre-installation requirements and post-installation verification.
Payment References	Payment processor reference information focusing on authorization and settlement using eFunds' ClearCommerce PaymentDirector™ software.
<i>Account Administrator (CSP) Guide</i>	How-to information about performing administration tasks.
Online Help for the Account Administrator (CSP) Tool	How-to information about performing configuration, management, and administration tasks using the Account Administrator (CSP) Tool. To access Help, click the Help link from any page of the Account Administrator Tool.

Documentation	Description
<i>ClearCommerce Engine API Reference and Guide</i>	Reference and how-to information about using the ClearCommerce Engine API to construct documents for sending information to and retrieving information from the ClearCommerce Engine.
<i>Document Hierarchy Reference</i>	Reference information about data that can be contained in ConfigDocs, OrderFormDocs, and ReportDocs. This is a three-volume set.
<i>Using the XML Input Component</i>	Reference and how-to information about using the XML input component.
Javadocs	Reference information about the Java classes that are available through the Java ClearCommerce Engine Application Programming Interface (API).

Each document is designed for specific types of users:

- Store administrators—responsible for managing an online store’s e-commerce activity; sometimes referred to as merchants.
- Risk managers—responsible for managing risk or monitoring potentially fraudulent activity for an online store.
- System administrators—responsible for administering all Engine-level activities, including ensuring successful communication with payment processors, and monitoring server performance.
- Account administrators—responsible for administering merchant account activities, including setting up merchants and managing user IDs and permissions. Frequently, a commerce service provider (CSP) acts as an account administrator.
- Integrators—responsible for integrating ClearCommerce software with applications from which the transactions originate, such as an online store’s electronic storefront or a call center or to other business systems, such as fulfillment, inventory, or accounting applications.

Online Resources

The following online resources are available.

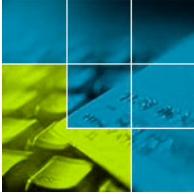
Developer Support Site

Additional documentation and tutorials are available on the Web. If you are a developer who is integrating the ClearCommerce Engine with an online store or other card-not-present application, be sure to visit eFunds Retail Solutions' ClearCommerce Software Developer Support Site:

`http://devsupport.clearcommerce.com`

You will need a user name and password (available from Customer Support) to log on.

About This Manual



Chapter 1

Introduction

To compliment the current C and Java APIs, ClearCommerce has created an XML interface to its Engine. An application developer can now choose between the C API, the Java API, or the XML input component to send transactions to the ClearCommerce Engine.

What is XML?

XML is short for *Extensible Markup Language*. XML is a meta-language used to define other languages. It is a markup language that specifies neither the tag set nor the grammar for that language. The *tag set* for a markup language defines the markup tags that have meaning to a language parser. For example, HTML has a strict set of tags that are allowed. You can use the tag `<TABLE>` but not the tag `<CHAIR>`. While the first tag has a specific meaning to an application using the data, and is used to signify the start of a table in HTML, the second tag has no specific meaning. That is because when HTML was defined, the tag set of the language was defined with it.

XML, by defining neither the tags nor the grammar, is completely extensible (thus its name). If you choose to use the tag `<TABLE>` and then nest within that tag several `<CHAIR>` tags, you may do so. If you wish to define a `TYPE` attribute for the `<CHAIR>` tag, you may do that as well.

Java and XML

It's hard to imagine two more complementary technologies: While the Java platform offers the foundation for shuttling code securely and portably around networks, XML technology can do the same for data,

offering a clean, platform-neutral way to represent content. Our focus will be on XML and Java, but XML is certainly not restricted to Java; XML can be used with other languages and technology.

The XML 1.0 standard was approved and published by the World Wide Web Consortium (W3C) on February 10, 1998. Since then, XML technology has quickly gained favor as a universal data interchange format for networked systems. Among the practical benefits are:

- Structure — to model data to any level of complexity
- Extensibility — to define new tags as needed
- Validation — to check data for structural correctness
- Media independence — to publish content in multiple formats
- Vendor and platform independence — to process any conforming document using standard commercial software or even simple text tools

Simple API for XML (SAX)

SAX is the Simple API for XML. It provides an event-based framework for parsing XML data, which is the process of reading through the document and breaking down the data into usable parts: At each step of the way, SAX defines events that can occur. For example, SAX defines an `org.XML.sax.ContentHandler` interface that defines methods such as **startDocument()** and **endElement()**. Implementing this interface allows complete control over these portions of the XML parsing process.

SAX is often mistaken for an XML parser. SAX only provides a framework for parsers to use and it defines events within the parsing process to monitor. A parser must be supplied to SAX to perform any XML parsing. This has resulted in many excellent parsers being made available in Java, such as Sun's Project X, the Apache Software Foundation's Xerces, Oracle's XML Parser, and IBM's XML4J. These can all be plugged into the SAX APIs and result in XML data. SAX APIs provide the means to parse a document, not the XML parser itself.

Document Object Model (DOM)

The Document Object Model (DOM), unlike SAX, has its origins in the World Wide Web Consortium (W3C). Whereas SAX is public-domain software, developed through long discussions on the XML-dev mailing list, DOM is a standard just as the actual XML specification itself is. The DOM is also not designed specifically for Java, but to represent the content and model of documents across all programming languages and tools. Bindings exist for JavaScript, Java, CORBA, and other languages, allowing the DOM to be a cross-platform and cross-language specification.

While SAX only provides access to the data within an XML document, DOM is designed to provide a means of manipulating that data. DOM provides a representation of an XML document as a tree. Because a tree is an age-old data representation, traversal and manipulation of tree structures are easy to accomplish in programming languages, Java being no exception. DOM also reads an entire XML document into memory, storing all the data in *nodes*, so the entire document is very fast to access; it is all in memory for the length of its existence in the DOM tree. Each node represents a piece of the data pulled from the original document.

Java API for XML Parsing (JAXP)

JAXP is Sun's Java API for XML Parsing. A relatively new addition to the XML developer's suite, it attempts to provide cohesiveness to the SAX and DOM APIs. While it does not compete with or replace either of these APIs, it does add some convenience methods to try to make the XML APIs easier to use for Java developers. It conforms to the SAX and DOM specifications, as well as adhering to the namespace recommendations. JAXP does not redefine SAX or DOM behavior, but ensures that all XML-conformant parsers can be accessed within Java applications through a standard pluggability layer.

XML is now part of the Sun JDK 1.3_02 Enterprise Edition. Sun now includes support for XML as part of the Enterprise Edition of JDK 1.3, version 2. This greatly simplifies setting up your environment to compile and run XML code.

HTTP POST

The Hypertext Transport Protocol (HTTP) is the language web clients and servers use to communicate with each other. It is essentially the backbone of the World Wide Web. HTTP is a simple, stateless protocol. A client, such as a web browser, makes a request, the web server responds, and the transaction is done. When the client sends a request, the first thing it specifies is an HTTP command, called a *method*, that tells the server the type of action it wants to perform. The most frequently used methods are GET and POST. The GET method is designed for getting information (a document, a chart, or the results from a database query), while the POST method is designed for posting information (a credit card number, some new chart data, or information that is to be stored in a database). For the purpose of this document, please consider HTTP to include both HTTP and HTTPS, secure HTTP

Java Secure Socket Extension (JSSE)

The Java™ Secure Socket Extension (JSSE) is a Java package that enables secure Internet communications. It implements a Java version of SSL (Secure Sockets Layer) and TLS (Transport Layer Security) protocols and includes functionality for data encryption, server authentication, message integrity, and optional client authentication. Using JSSE, developers can provide for the secure passage of data between a client and a server running any application protocol (such as HTTP, Telnet, NNTP, and FTP) over TCP/IP.

XML and the ClearCommerce Engine

To complement the current C and Java APIs, ClearCommerce Engine now has an XML interface. An application developer can choose between Java, C, or XML to send transactions to the ClearCommerce Engine.

XML documents can be sent to the XML input component, which is part of the ClearCommerce Engine, using the HTTP POST method. The HTTP POST method can be invoked from a browser, Java code, or other programming languages. Included in this document are examples of POSTing an XML document from a browser and Java code.

To create an XML document and send it to the XML input component, no client software is required from ClearCommerce. An understanding of the ClearCommerce document hierarchy will be essential in creating a correctly formed XML document bound for the ClearCommerce Engine. (See *Document Hierarchy Reference: OrderFormDocs* for more information.) To create an XML document, a developer can use an XML editor, text editor, Java code or other mechanisms that are capable of creating XML documents. To transport an XML document to the XML input component, the client can utilize a browser, Java code or other tools capable of doing HTTP POSTs. To aid in the development of XML input component client applications, sample Java code (see Chapter 3) and sample XML documents (See Chapter 2) are provided.

Validating Your XML Using the OrderFormDoc DTD

A DTD (Document Type Definition) document is a document separate from your XML document that contains formal definitions of all of the data elements in the XML document.

When creating XML, you should ensure that your XML is both “well formed” and “valid.” A *well-formed* XML document is a document that conforms to XML syntax rules, whereas a *valid* XML document is a well-formed document that also conforms to the rules contained in a DTD.

An OrderFormDoc DTD is available for download from the ClearCommerce Developer Support Site:

```
http://devsupport.clearcommerce.com/resources/
```

Comments are included in the body of the DTD to help you validate your OrderFormDoc XML documents.

XML Web Sites and Tools

The following resources and Web sites are available to help you learn more about using XML.

XML Web Sites

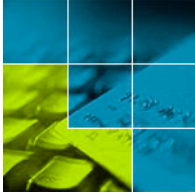
- <http://www.xml.org> - XML Portal Web site

- <http://www.w3.org> - World Wide Web Consortium Web site
- <http://www.XMLspy.com> - XML editor

Java and XML Tools

These are some recommended tools that are in use with XML:

- Java 2 Standard Edition JDK 1.2.2 or 1.3 with Java Secure Sockets Extension (JSSE) 1.0.3_02
- OR-
- Java 2 Standard Edition JDK 1.4.1 with the bundled JSSE
- JAXP 1.1 (included in JDK 1.3_02 and later)



Chapter 2

XML Interface

XML Input to the ClearCommerce Engine

The XML input component and client are similar to their C and Java counterparts. From the XML client application, you can send XML documents to the XML input component using a POST method. The XML input component can be configured for multiple ports; each of which can be either HTTPS (secure) or HTTP (clear).

The XML input component is configured with two ports. Normally one of the ports is configured for secure transactions (Port 11500) and the other for clear transactions (11501). The clear port is often used for testing, but can be used for production when the application is behind a secure firewall.

Here is a sample flow of data through the system:

- An XML document is created in the user's application. In this example, we will assume the XML document was created in Java using JAXP. This book includes examples of Java client code (see Sample Java Code).
- The XML document is then transported to the XML input component. Java contains utilities in the `java.net` package to open an URL connection and POST the document to the XML input component. In this example, the XML document is an HTTPS POST to port 11500. The data can be sent secure, HTTPS, or not secure, HTTP. The port number of the XML Component is configurable.
- The XML input component receives the XML document and converts it to an internal Engine representation of a transaction. The XML input component then passes the transaction to the Director for routing and processing.

- Once the transaction's routing is complete the response is returned to the XML input component, which converts the transaction back to XML and returns it to the application.
- The application evaluates the results and performs the appropriate actions.

This process is illustrated in the following diagram:

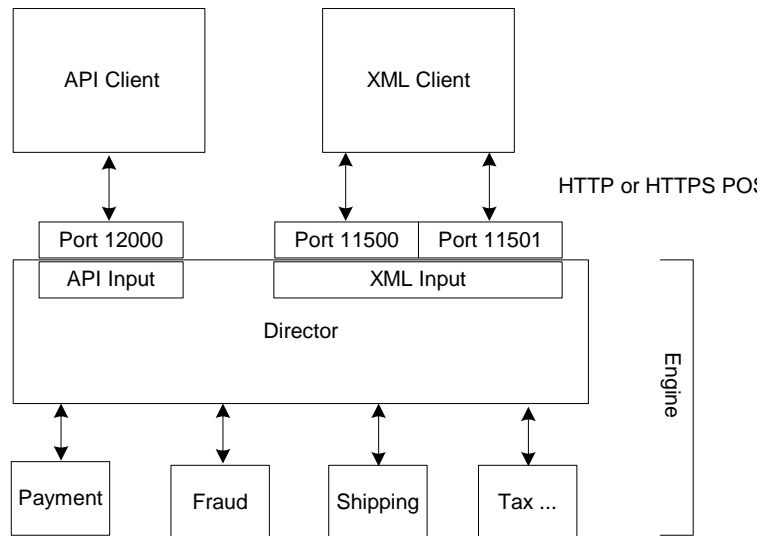


Figure 2-1 XML Input to the Engine

XML and the ClearCommerce API Document Hierarchy

The XML documents bound for the ClearCommerce Engine are designed after the API document hierarchy. There is a direct mapping from the API document hierarchy, described in the *ClearCommerce Engine API Reference and Guide*, to the XML documents. Once there is familiarity with the document hierarchy, developers can generate XML or API documents. The only difference being the syntax of the document and the methods used to create it. The following table shows the cross reference between the API document hierarchy and XML:

Table 2-1 API to XML Cross Reference

API Term	XML Term	Description and Mapping
Document	Document	<p>A <i>document</i> is a container that organizes hierarchical information.</p> <p>Mapping: There is a direct mapping between an API <i>document</i> and an XML <i>document</i></p>
<p>Record</p> <p>Example <User>...</User></p>	<p>Element</p> <p>Example <User>...</User></p>	<p>A <i>record</i> is a container that groups related information into a logical unit.</p> <p>An XML <i>element</i> is delimited by tags and can enclose other elements.</p> <p>Mapping: A <i>record</i> maps directly to an <i>element</i> with no element text.</p>
<p>Field and Value</p> <p>Example Name = XML_store</p>	<p>Element with Element text</p> <p>Example <Name>XML_store</Name></p>	<p><i>Fields</i> can be considered to be attributes of a document. Each <i>field</i> can be assigned a specific type of value, such as string, integer, currency, or timestamp. <i>Fields</i> can be considered as leaf nodes of the document tree structure. They cannot contain subfields.</p> <p>An <i>element</i> can have text within its tags. <i>Element Text</i> is the text between the <i>element</i> tags. For example, in the element <Name>XML_store </Name>. XML_store is the element text.</p> <p>Mapping: A <i>field</i> and <i>value</i> map directly to an <i>element</i> and <i>element text</i> with the field's value as the element's text.</p> <p>See note in Sample XML Documents</p>

Table 2-1 API to XML Cross Reference (Continued)

API Term	XML Term	Description and Mapping
<p>Data Type</p> <p>Example1 <i>ClientId="4"</i></p> <p>Example2 <i>Total= 35430</i></p> <p>Note there is an implied attribute of <i>Currency 840</i> to the Total value. It is assigned using an API method call.</p>	<p>Attribute and Value pairs</p> <p>Example1 – 1 Attribute <i><ClientId DataType="S32">2 08 </ClientId></i></p> <p>Example2 – 1 Attribute <i><Total DataType="Money" Currency="840">35 430 </Total></i></p>	<p>All <i>fields</i> in a <i>record</i> or <i>document</i> are of a specific <i>data type</i>. A <i>data type</i> describes the <i>field</i>. The API <i>data types</i> are the following:</p> <ul style="list-style-type: none"> • DateTime • ExpirationDate • Money • Numeric • S32 • StartDate • String <p>XML input component also supports the following attribute/value pairs:</p> <ul style="list-style-type: none"> • Locale="value" • Currency="value" • Precision="value" • xml:space="value"

Note: The XML input component also supports CDATA sections and character references and substitutions.

Using the definitions of the API and XML documents and how they map, a sample document can be created. The following document contains the *User* section from an XML and API document. These sections are equivalent and mean the same thing to the ClearCommerce Engine. Notice that the *element* (XML) and *record* (API) names (User, Name, Password and ClientId) are the same. The field *Name* with value *XML_store* is the same as an element and element text.

XML Document

```

<User> (Element User with no element text)
  <Name>XML_store</Name> (Element Name with text
XML_store)
  <Password>xml</Password> (Element Password with
text XML)
  <ClientId (Element ClientId)
    DataType="S32"> (DataType is an attribute with
value S32)
    208 (ClientId has a value of 208 of S32)
  </ClientId> (Close element ClientID)
</User> (Close element User)

```

API Document

```

User // Record User
  Name = XML_store //Field Name with value XML_store
  Password = xml //Field Password with value XML
  ClientId = 208 //Field ClientId with value 208 and
DataType of S32

```

Sample XML Document

An example PreAuth XML document appears on page 13. The PreAuth transaction obtains authorization from the issuing authority for the amount of the cardholder's purchase. An Approved PreAuth places a hold on the

“open-to-buy” amount of a cardholder’s account. Before a merchant can begin the process to collect money, the order must be completed by a corresponding PostAuth transaction.

Note: Since element text is everything between the end of the beginning tag and the start of the ending tag, be careful with white space and other non-printable characters like tabs and carriage returns in this area. The Engine may reject or ignore the document because the field is not syntactically correct due to the white space characters. For example,

```
<Pipeline> PaymentNoFraud</Pipeline>
```

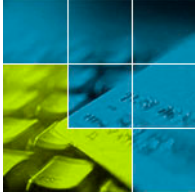
is actually the text string “<space>PaymentNoFraud”. This element text will have the <space> stripped by the XML input component, but this is not a good practice for building XML documents. For readability, the developer may break a line within the element text. This could also cause the element text to be invalid.

PreAuth XML Example

```

<?XML version="1.0" encoding="UTF-8"?>
<EngineDocList>
  <DocVersion>1.0</DocVersion>
  <EngineDoc>
    <ContentType>OrderFormDoc</ContentType>
    <User>
      <Name>XML_store</Name>
      <Password>xml</Password>
      <ClientId DataType="S32">208</ClientId>
    </User>
    <Instructions>
      <Pipeline>PaymentNoFraud</Pipeline>
    </Instructions>
    <OrderFormDoc>
      <Mode>Y</Mode>
      <Consumer>
        <PaymentMech>
          <CreditCard>
            <Number>4111111111111111</Number>
            <Expires DataType="ExpirationDate"
              Locale="840">01/04
            </Expires>
          </CreditCard>
        </PaymentMech>
      </Consumer>
      <Transaction>
        <Type>PreAuth</Type>
        <CurrentTotals>
          <Totals>
            <Total DataType="Money" Currency="840">
              35430 </Total>
            </Totals>
          </CurrentTotals>
        </Transaction>
      </OrderFormDoc>
    </EngineDoc>
  </EngineDocList>

```

Chapter 3

Sample Client Applications

The samples discussed in this chapter are available to assist in building simple client applications that generate and send XML documents to the XML input component. These are not all of the methods available, but they will give the basics needed to create and send XML documents.

Sample HTML Client

The following is sample HTML code that sends an XML document to the ClearCommerce Engine:

```
<HTML>
<HEAD>

  <TITLE>XML Input Component Post Test</TITLE>

<SCRIPT language="JavaScript">
<!--
function validate(){

  var data = document.main_form.CLRCMRC_XML.value;

  if(data.length == 0){
    alert("Error: There is no XML input data
specified!");
    return false;
  }

  //Add your validation code here //

  //////////////////////////////////////
```

```

var outdata = "";
var dataok = false;

//entity encode non-printable chars
var EIGHT_BIT_MASK = 128;
var i=0;
var c = 0;
for(i=0;i<data.length;i++){
    var c = data.charAt(i);
    var n = ascii_value(c);
    //we do not encode \n and \r for readability sake
    if(n & EIGHT_BIT_MASK || (n < 32 && n != 10 && n
    != 13)){
        outdata += "&#" + n + ";";
    }else{
        outdata += data.charAt(i);
    }
}

document.main_form.CLRCMRC_XML.value = outdata;
return dataok;
}

//This is inefficient but it is necessary to guard
against
//non-printable characters. If you know of a more
efficient
//way to do it, go ahead.
function ascii_value (c)
{
    //loop through all possible ASCII values
    var i;
    for (i = 0; i < 256; ++ i){
        var h = i . toString (16);
        if (h . length == 1){
            h = "0" + h;
        }
        // insert a % character into the string
        h = "%" + h;
    }
}

```

Sample Client Applications

```
        // determine the character represented by the
escape code
        h = unescape (h);

        // if the characters match, we've found the ASCII
value
        if (h == c){
            break;
        }
    }
    return i;
}

-->
</SCRIPT>
</HEAD>
<BODY BGCOLOR="#FFFFFF">

<FONT SIZE=5> XML Input Component Verification Test
</FONT>
<P>
<HR ALIGN=LEFT WIDTH=100% SIZE=2>

Enter the XML form that you want sent to the XML Input
Component and press the Submit button.
This form will be POST'd to the XML Input Component.

<!--
Modify this line to point to the server and servlet
name on the system being tested
-->

<form NAME="main_form" ACTION="http://<ClearCommerce
Engine server name here>:11501" METHOD=POST
OnSubmit="return validate();">

<TextArea cols="80" rows="40" name="CLRCMRC_XML"
></TEXTAREA>
```


Sample Java Clients

Read XML document from a file and send to ClearCommerce Engine

This sample code uses a Java file (*XMLtoEngineFromFile.java*) described below and it demonstrates the following:

- Reads an XML document from a file
- POST the document to the XML input component
- Read the results from the XML input component

To use this sample, do the following:

- 1 Use the sample run.bat and make.bat code described below to build similar batch files
- 2 Set up the correct paths in run.bat and make.bat for your Java code
- 3 Compile the file
> make.bat XMLtoEngineFromFile.java
- 4 Modify the test XML file with the appropriate user, password and clientId
- 5 Run the file
> run.bat XMLtoEngineFromFile "http://test4x.clearcommerce.com:11501"
"C:\XML Component\XML\PreAuth.XML"
- 6 Review the results

run.bat File Example

```
@setlocal&set a=@&set e=@&
:MAIN
%a% REM Please enter the path to where you have
installed Java
%a% set JAVA_HOME=C:\jdk1.3\lib
%a% REM Please enter the path to where JAXP is
installed
%a% set JAXP_HOME=C:\Program Files\JavaSoft\jaxp-1.1
%a% set
LOCALCLASSPATH=.;%JAXP_HOME%\jaxp.jar;%JAXP_HOME%\cr
imson.jar;%JAXP_HOME%\xalan.jar
echo LocalPath=%LOCALCLASSPATH%
java -classpath "%LOCALCLASSPATH%" %1 %2 %3 %4
```

make.bat File Example

```
@setlocal&set a=@&set e=@&
:MAIN
%a% REM Please enter the path to where you have
installed Java
%a% set JAVA_HOME=C:\jdk1.3\lib
%a% REM Please enter the path to where JAXP is
installed
%a% set JAXP_HOME=C:\Program Files\JavaSoft\jaxp-1.1
%a% set
LOCALCLASSPATH=.;%JAXP_HOME%\jaxp.jar;%JAXP_HOME%\cr
imson.jar;%JAXP_HOME%\xalan.jar
javac -classpath "%LOCALCLASSPATH%" %1
```

XMLtoEngineFromFile.java Example

```
/**
 * Title: XMLtoEngineFromFile
 * Description: A CCE XML Input Component Client
class, with a
 * main() to show the simple case usage.
 *
 * To display usage information compile and run:
 * javac XMLtoEngineFromFile.java
 * java XMLtoEngineFromFile
 *
 * Copyright: ClearCommerce Corp Copyright (c) 2004
 * @author: Brian Kamery
 * @version 1.2
 */

public class XMLtoEngineFromFile
{
    private URL m_url;

    /**
     * Constructor
     * @param cceurl URL that defines the CCE XML input
component,
     * (either secure or non-secure), typically HTTP
ot HTTPS.
     */
    public XMLtoEngineFromFile(URL cceurl)
    {
        m_url = cceurl;
    }

    /**
     * Access ot the EngineUrl for this Client object
     * @returns cceurl URL that represents the location
of the targeted
     * XML inputComponent of the CCE
     */
    public URL getEngineUrl()
    {
        return m_url;
    }
}
```

```

    }

    /**
     * Modifier Access of the EngineUrl for this Client
     object
     * @param cceurl URL that represents the location
     of the targeted
     * XML inputComponent of the CCE
     */
    public void setEngineUrl(URL cceurl)
    {
        m_url = cceurl;
    }

    /**
     * Sends a request to the engine that is stored in
     the specified file.
     * This could easily be modified to send a String
     or byte array or whatever.
     * @param input File that is a valid CCE XML request
     to be sent
     * @throws IOException when any IO problem is
     encountered
     */
    public String sendRequest(File input)
        throws IOException
    {
        InputStream filein = new
        FileInputStream(input);

        URLConnection conn = m_url.openConnection();
        conn.setRequestProperty("Content-Type",
        "application/xml");

        conn.setDoOutput(true);

        OutputStream cceout = conn.getOutputStream();

        //No longer necessary as of 5.5 - though it is
        still OK to POST as formdata
        //cceout.write("CCE_XML=".getBytes());
        //we need to loop through the incoming data
        checking for 8-bit character
        //and encode them as XML Entities when they

```

```

are found. This really should be
    //done for all non-printable characters,
    according to the XML Spec, but the
    //CCE is flexible enough to handles all the 7-
    bit characters
    byte[] data = new byte[2048];
    int k = filein.read(data);
    int MASK8 = 128;
    while (k > 0)
    {
        int start = 0;
        for (int i = 0; i < k; i++)
        {
            if ((data[i] & MASK8) != 0)
            {
                data[i] ^= MASK8;

                int value = (int)(data[i] ^ MASK8);
                cceout.write(data, start, i);

                String encoded = "&#" + value + ";";
                cceout.write(encoded.getBytes());
                System.out.print(encoded);
                start = i + 1;
            }
        }

        cceout.write(data, start, k - start);
        k = filein.read(data);
    }

    //end of while
    filein.close();
    cceout.close();

    StringBuffer response = new StringBuffer();
    BufferedReader bin = new BufferedReader(new
    InputStreamReader(
        conn.getInputStream()));
    String line = bin.readLine();
    while (line != null)
    {

```

```

        response.append(line);
        line = bin.readLine();
    }

    bin.close();

    return response.toString();
}

/**
 * A Utility method to print the commandline usage
of this class
 */
private static void printUsage()
{
    System.out.println("Usage: XMLtoEngineFromFile
EngineURL XmlFilename");
}

/**
 * A Simple and minimal use of the
XMLtoEngineFromFile
 */
public static void main(String[] args)
{
    if (args.length != 2)
    {
        printUsage();
        System.exit(0);
    }

    try
    {
        //Setup JSSE for pre- JDK1.4 JRE's .
        //Note: For pre JDK1.4 runtime you must
have the JSSE jars in the CLASSPATH!

//System.setProperty("java.protocol.handler.pkgs",
//
"com.sun.net.ssl.internal.www.protocol");
        // Security.addProvider(new
com.sun.net.ssl.internal.ssl.Provider());
        //create the client Object on the specified

```

```
URL
    URL url = new URL(args[0]);
    XMLtoEngineFromFile ccexml = new
XMLtoEngineFromFile(url);

    //check that the specified file exists
    File file = new File(args[1]);
    if (!file.exists())
    {
        System.err.println("File does not exist:
"
            + file.getAbsolutePath());
    }

    //send the request and get the response
    String response = ccexml.sendRequest(file);
    System.out.println(response);
}
catch (MalformedURLException e)
{
    System.err.println("Malformed URL: " +
args[0]);
    System.err.println();
    printUsage();
}
catch (Exception e)
{
    e.printStackTrace();
    printUsage();
}
}
```

Create an XML document with JAXP and send to ClearCommerce Engine

This sample code uses a Java file (`XMLtoEngine.java`) and an input file (`input.txt`), both described below. It demonstrates the following:

- Creates an XML document in Java
- POST the document to the XML input component

- Read the results from the XML input component

To use this code do the following:

- 1 Use the sample `run.bat` and `make.bat` code described above to build similar batch files.
- 2 Set up the correct paths in `run.bat` and `make.bat` for your Java code.
- 3 Compile the file.


```
> make.bat XMLtoEngine.java
```
- 4 Modify the `input.txt` file with the appropriate parameters.
- 5 Run the file.


```
> run.bat XMLtoEngine input.txt
```
- 6 Review the results

The sample `XMLtoEngine.java` reads in a file, like `input.txt` described below, and populates the document and then interprets the results.

`Input.txt` contains the following information:

Table 3-1

Line	Name	Description	Example
1	UserName	User name performing the transaction	XML_store
2	User Password	Password for the user	xml
3	ClientId	The entity's ID	208
4	Credit Card	Sample credit card number	4111111111111111
5	Expiration Date	Expiration date of the credit card Form: mm/yy or mm/dd/yy	05/02
6	Auth Type	Type of transaction, either Auth or PreAuth	PreAuth
7	Total	Amount of money to authorize in pennies	12367

Table 3-1 (Continued)

Line	Name	Description	Example
8	Host	Host of XML input component	test4x.clearcommerce.com
9	Port	Port XML input component is listening on	11501
10	bCrypto	Send transaction HTTP (false) or HTTPS (true)	false

Sample input.txt File

The following example illustrates the contents of an `input.txt` file:

```
XML_store
xml
208
41111111111111111111
05/02
PreAuth
12367
test4x.clearcommerce.com
11501
false
```

Sample XMLtoEngine.java File

```

/**
 * Title: XMLtoEngine
 * Description: A CCE XML Input Component Client
class, with a
 * main() to show analysis of response document.
 *
 * To display usage information compile and run:
 * javac XMLtoEngine.java
 * java XMLtoEngine
 *
 * Copyright: ClearCommerce Corp Copyright (c) 2004
 * @author: Brian Kamery
 * @version 1.2
 */

import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.FactoryConfigurationError;
import
javax.xml.parsers.ParserConfigurationException;

public class XMLtoEngine
{
    // Global value so it can be ref'd by the tree-
adapter
    public XMLtoEngine()
    {
    }

    public Document createRequestDocument(Properties
props)
    {
        String userName =
props.getProperty("Username");
        String userPassword =
props.getProperty("Password");
        String clientId =
props.getProperty("ClientId");
        String creditCardNum =
props.getProperty("CardNumber");
        String creditCardExpiration =
props.getProperty("CardExpDate");
    }
}

```

```
        String authType =
        props.getProperty("TransactionType");
        String total = props.getProperty("TransTotal");
        String hosturl =
        props.getProperty("EngineUrl");
        String strPipeline =
        props.getProperty("Pipeline");

        DocumentBuilderFactory factory =
        DocumentBuilderFactory.newInstance();
        try
        {
            DocumentBuilder builder =
            factory.newDocumentBuilder();
            Document document = builder.newDocument();

            Node pcdData = null;
            Element element = null;

            //
            // Create the DOM
            //
            // Create EngineDocList
            Element EngineDocList =
            document.createElement("EngineDocList");
            document.appendChild(EngineDocList);

            // Create DocVersion
            Element DocVersion =
            document.createElement("DocVersion");
            EngineDocList.appendChild(DocVersion);
            pcdData = document.createTextNode("1.0");
            DocVersion.appendChild(pcdData);

            // Create EngineDoc
            Element EngineDoc =
            document.createElement("EngineDoc");
            EngineDocList.appendChild(EngineDoc);

            // Create ContentType
            Element ContentType =
            document.createElement("ContentType");
            EngineDoc.appendChild(ContentType);
```

```

        pcdData =
document.createTextNode("OrderFormDoc");
        ContentType.appendChild(pcdData);

        // Create User
        Element User =
document.createElement("User");
        EngineDoc.appendChild(User);

        // Create Name
        Element Name =
document.createElement("Name");
        User.appendChild(Name);
        pcdData = document.createTextNode(userName);
        Name.appendChild(pcdData);

        // Create Password
        Element Password =
document.createElement("Password");
        User.appendChild(Password);
        pcdData =
document.createTextNode(userPassword);
        Password.appendChild(pcdData);

        // Create ClientId
        Element ClientId =
document.createElement("ClientId");
        User.appendChild(ClientId);
        pcdData = document.createTextNode(clientId);
        ClientId.appendChild(pcdData);
        ClientId.setAttribute("DataType", "S32");

        // Create Instructions
        Element Instructions =
document.createElement("Instructions");
        EngineDoc.appendChild(Instructions);

        // Create Pipeline
        Element Pipeline =
document.createElement("Pipeline");
        Instructions.appendChild(Pipeline);
        pcdData =
document.createTextNode(strPipeline);

```

```
        Pipeline.appendChild(pCDATA);

        // Create OrderFormDoc
        Element OrderFormDoc =
document.createElement("OrderFormDoc");
        EngineDoc.appendChild(OrderFormDoc);

        // Create Mode
        Element Mode =
document.createElement("Mode");
        OrderFormDoc.appendChild(Mode);
        pCDATA = document.createTextNode("Y");
        Mode.appendChild(pCDATA);

        // Create Consumer
        Element Consumer =
document.createElement("Consumer");
        OrderFormDoc.appendChild(Consumer);

        // Create PaymentMech
        Element PaymentMech =
document.createElement("PaymentMech");
        Consumer.appendChild(PaymentMech);

        // Create CreditCard
        Element CreditCard =
document.createElement("CreditCard");
        PaymentMech.appendChild(CreditCard);

        // Create Number
        Element Number =
document.createElement("Number");
        CreditCard.appendChild(Number);
        pCDATA =
document.createTextNode(creditCardNum);
        Number.appendChild(pCDATA);

        // Create Expires
        Element Expires =
document.createElement("Expires");
        CreditCard.appendChild(Expires);
        pCDATA =
document.createTextNode(creditCardExpiration);
```

```

        Expires.appendChild(pCDATA);
        Expires.setAttribute("DataType",
"ExpirationDate");
        Expires.setAttribute("Locale", "840");

        // Create Transaction
        Element Transaction =
document.createElement("Transaction");
        OrderFormDoc.appendChild(Transaction);

        // Create Type
        Element Type =
document.createElement("Type");
        Transaction.appendChild(Type);
        pCDATA = document.createTextNode(authType);
        Type.appendChild(pCDATA);

        // Create CurrentTotals
        Element CurrentTotals =
document.createElement("CurrentTotals");
        Transaction.appendChild(CurrentTotals);

        // Create Totals
        Element Totals =
document.createElement("Totals");
        CurrentTotals.appendChild(Totals);

        // Create Total
        Element Total =
document.createElement("Total");
        Totals.appendChild(Total);
        pCDATA = document.createTextNode(total);
        Total.appendChild(pCDATA);
        Total.setAttribute("DataType", "Money");
        Total.setAttribute("Currency", "840");

        return document;
    }
    catch (ParserConfigurationException pce)
    {
        // Parser with specified options can't be
built
        pce.printStackTrace();
    }
}

```

```

    }

    return null;
}

// The processing of the document, records, and
fields
public Document send(
    Document request,
    String hosturl)
{

    Document returnDocument = null;

    //
    // For https you need to have the JSSE package
from sun and
    // execute the following code.
    //

//System.setProperty("java.protocol.handler.pkgs",
    //    "com.sun.net.ssl.internal.www.protocol");
    //Security.addProvider(new
com.sun.net.ssl.internal.ssl.Provider());
    try
    {
        DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
        DocumentBuilder builder =
factory.newDocumentBuilder();

        // first build the base structure
        // by getting a builder and instantiating
a new document
        // These classes are from the sun.xml.parsers
package.

        //
        //
        // Setup URL and URLConnection
        java.net.URL url = new
java.net.URL(hosturl);

        // Used for testing https service

```

```

        //java.net.URL url = new
java.net.URL("https://www.verisign.com/");
        java.net.URLConnection connection =
url.openConnection();
        connection.setRequestProperty("Content-
Type", "application/xml");
        connection.setDoOutput(true);

        // Connect to outStream
        java.io.OutputStream outStream =
connection.getOutputStream();

        // Send Document to XML Component
        PrintStream pout = new
PrintStream(outStream);

        //no longer needed for CCE 5.5+
        pout.print("CLRCMRC_XML=");

        printNode(pout, request, "", "", "");

        java.io.InputStream inputStream =
connection.getInputStream();
        // Get the return information from XML
Component
        returnDocument = builder.parse(inputStream);

        return returnDocument;
    }
    catch (SAXParseException spe)
    {
        // Error generated by the parser
        System.out.println("\n** Parsing error" +
", line "
            + spe.getLineNumber() + ", uri " +
spe.getSystemId());
        System.out.println("    " +
spe.getMessage());

        // Use the contained exception, if any
Exception x = spe;
        if (spe.getException() != null)

```

```

        {
            x = spe.getException();
        }

        x.printStackTrace();
    }
    catch (SAXException sxe)
    {
        // Error generated by this application
        // (or a parser-initialization error)
        Exception x = sxe;
        if (sxe.getException() != null)
        {
            x = sxe.getException();
        }

        x.printStackTrace();
    }
    catch (ParserConfigurationException pce)
    {
        // Parser with specified options can't be
built
        pce.printStackTrace();
    }
    catch (IOException ioe)
    {
        // I/O error
        ioe.printStackTrace();
    }

    return returnDocument;
}

private void analyze(Document responseDoc)
{
    String transactionStatus =
getValueByPath("EngineDoc.Overview.TransactionStatus",
                responseDoc.getDocumentElement());

    if ("A".equals(transactionStatus))
    {

```



```

        String ccReturnMsg =
getValueByPath("EngineDoc.OrderFormDoc.Transaction.C
cReturnMsg", returnDocument.getDocumentElement());
        //most likely ccReturnMsg is null!!!
        System.out.println(">>>> Transaction
Error:\nCcReturnMsg="+ccReturnMsg + "\nSee
MessageList for details.");
        /**@todo YOUR BUSINESS LOGIC for Error case
HERE*/
        }else if(ccErrCode.equals("1")){
        //transactionStatus = "A"
        String authcode =
getValueByPath("EngineDoc.OrderFormDoc.Transaction.A
uthCode", returnDocument.getDocumentElement());
        String transId =
getValueByPath("EngineDoc.OrderFormDoc.Transaction.I
d", returnDocument.getDocumentElement());

        System.out.println(">>>> Transaction
Approved\nAuthCode="+authcode+"\nTransactionId=" +
transId);

        /**@todo YOUR BUSINESS LOGIC for Approved
case HERE*/
        }else if(ccErrCode.equals("500") ||
ccErrCode.equals("501") || ccErrCode.equals("502") ){
        //transactionStatus = "F"
        Element fraudInfo =
getElementByPath("EngineDoc.OrderFormDoc.FraudInfo",
returnDocument.getDocumentElement());
        System.out.println(">>>> Transaction
Fraudulent\nSee FraudInfo.Alerts for more detail\n");
        if(fraudInfo != null){
            printNode(System.out, fraudInfo, "", "
", "\n");
        }

        /**@todo YOUR BUSINESS LOGIC for Error case
HERE*/
        }else{
        //transactionStatus = "D"
        String ccReturnMsg =
getValueByPath("EngineDoc.OrderFormDoc.Transaction.C
cReturnMsg", returnDocument.getDocumentElement());

        System.out.println(">>>> Transaction

```

```

Declined\nCcReturnMsg="+ccReturnMsg + "\nSee
MessageList for details.");

        /**@todo YOUR BUSINESS LOGIC for Declined
case HERE*/
        }

    }

    //
    // What are we expecting as inputs
    //
    public static void showUsage()
    {
        System.err.println("Usage: java XMLtoEngine
<Properties File>");
        System.out.println("The input file is a Java
Properties file, (Format: \"name=value\")\n"
+ "Here is a sample file, it must contain
all of the following values:\n"
+ "Username=user1\n"
+ "Password=password\n"
+ "ClientId=12345\n"
+ "CardNumber=4111111111111111\n"
+ "CardExpDate=01/07\n"
+ "TransactionType=PreAuth\n"
+ "TransTotal=1286\n"
+ "EngineUrl=https://localhost:11500\n"
+ "Pipeline=PaymentNoFraud\n");
    }

    // The program
    public static void main(String[] args)
    {
        XMLtoEngine xmlclient = new XMLtoEngine();

        // Read the input values
        try
        {
            Properties props = new Properties();
            props.load(new FileInputStream(args[0]));
            System.out.println("Executing example with
the following values:");

```



```

        if(contentType.equals("ConfigDoc") ||
contentType.equals("ReportDoc")){
            String maxsevStr =
getValueByPath("EngineDoc.MessageList.MaxSev",
responseDoc.getDocumentElement());
            int maxsev = (maxsevStr == null) ? 0 :
Integer.parseInt(maxsevStr);
            if(maxsev == 0){
                System.out.println(">>>> " +
contentType + " request successful." );
                /**@todo Your business logic to handle
a ConfigDoc or ReportDoc success here */
            }else if(maxsev >= 5){
                //Emergency = 8, Alert = 7, Critical
= 6, Error = 5,
                System.out.println(">>>> There was
an error processing your " + contentType);
                /**@todo Your business logic to handle
a ConfigDoc or ReportDoc error here */
            }else {
                //Warning = 4, Notice = 3, Info = 2,
Debug = 1,
                System.out.println(">>>> " + contentType
+ " request successful, but there was a warning." );
                /**@todo Your business logic to handle
a ConfigDoc or ReportDoc success with warning or other
notice */
            }
        }else{

            // System.out.println("Root element ="
+ root.getTagName());
            String transactionStatus =
getValueByPath("EngineDoc.Overview.TransactionStatus
",
                responseDoc.getDocumentElement());
            if (transactionStatus == null)
            {
                //The response has not
Overview.TransactionStatus, thus (unless a huge
error)
                // no Overview element. Therefore, one
of the following cases is true:
                //(1) The CCE is a pre CCE5.5
                //(2)the PostProcessor is turned off

```

```

xmlclient.analyzeResponsePre5_5(responseDoc);
        }
        else
        {
            xmlclient.analyze(responseDoc);
        }
    }
}
catch (Exception io)
{
    io.printStackTrace();
    System.out.println("Please check your
input:");
    showUsage();
    System.exit(-1);
}
}

// main
private static Element getElementByPath(
    String path,
    Element elem)
{
    String tagname = path;
    String nextpath = null;
    int k = path.indexOf('.');
    if (k > 0)
    {
        tagname = path.substring(0, k);
        nextpath = path.substring(k + 1);
    }

    Element child = null;

    NodeList list = elem.getChildNodes();
    for (int i = 0; i < list.getLength(); i++)
    {
        Node node = list.item(i);
        if (node.getNodeType() == node.ELEMENT_NODE)
        {
            child = (Element)node;

```

```

        if (tagname.equals(child.getTag_name()))
        {
            break;
        }
        else
        {
            child = null;
        }
    }
}

if (child == null)
{
    return null;
}
else
{
    if (nextpath != null)
    {
        return getElementByPath(nextpath, child);
    }
    else
    {
        return child;
    }
}
}

private static String getValueByPath(
    String path,
    Element elem)
{
    Element child = getElementByPath(path, elem);

    if (child != null)
    {
        NodeList list = child.getChildNodes();
        for (int i = 0; i < list.getLength(); i++)
        {
            Node node = list.item(i);
            if (node.getNodeType() == node.TEXT_NODE)

```

```

        {
            return node.getNodeValue();
        }
    }
}

return null;
}

/**
 * <p>
 * This will print a DOM <code>Node</code> and then
recurse
 * on its children.
 * </p>
 *
 * @param node <code>Node</code> object to print.
 * @param indent <code>String</code> spacing to
insert
 *          before this <code>Node</code>
 */
/**
 * Returns true if the string s constitutes white
space
 * Creation date: (5/4/2001 10:39:21 AM)
 * @return boolean
 * @param s java.lang.String
 */
public static boolean whiteSpace(String s)
{
    //System.out.println("[ "+s+" ]");
    int length = s.length();
    for (int i = 0; i < length; i++)
    {
        char c = s.charAt(i);
        if ((c != ' ') && (c != '\n') && (c != '\t'))
        {
            return false;
        }
    }

    return true;
}

```

```

    }

    /**
     * <p>
     * This will print a DOM <code>Node</code> and then
recurse
     * on its children.
     * </p>
     *
     * @param node <code>Node</code> object to print.
     * @param indent <code>String</code> spacing to
insert
     *          before this <code>Node</code>
     *
     * printNode was created to replace XmlDocument
was dropped in the JAXP 1.1 release.
     */
    public static void printNode(
        java.io.PrintStream outputPrintStream,
        Node node,
        String iBase,
        String iInc,
        String lineSep)
    {
        // Determine action based on node type
        boolean elementTextFound = false;

        //node.normalize();
        switch (node.getNodeType())
        {
        case Node.DOCUMENT_NODE:

            // recurse on each child
            NodeList nodes = node.getChildNodes();
            if (nodes != null)
            {
                for (int i = 0; i < nodes.getLength(); i++)
                {
                    printNode(outputPrintStream,
nodes.item(i), iBase, iInc,
                            lineSep);
                }
            }
        }
    }

```

```

    }

    break;

    case Node.ELEMENT_NODE:

        String name = node.getNodeName();
        outputPrintStream.print(lineSep + iBase +
"<" + name);

        NamedNodeMap attributes =
node.getAttributes();
        for (int i = 0; i <
attributes.getLength(); i++)
        {
            Node current = attributes.item(i);
            outputPrintStream.print(" " +
current.getNodeName() + "=\""
                + current.getNodeValue() + "\"");
        }

        // recurse on each child
        NodeList children = node.getChildNodes();
        if (children != null)
        {
            outputPrintStream.print(">");
            for (int i = 0; i <
children.getLength(); i++)
            {
                // We need to check to see if there
is a text node
                // If there is then the text needs
to be printed on the same line
                // as the element tags.
                if (((children.item(i)).getNodeName()
== Node.TEXT_NODE)
                    &&
!(((children.item(i)).getNodeValue()).equals(" ")))
                {
                    elementTextFound = true;
                }

                printNode(outputPrintStream,
children.item(i),

```

```

        iBase + iInc, iInc, lineSep);
    }

    if (elementTextFound)
    {
        outputPrintStream.print("</" + name
+ ">");
    }
    else
    {
        outputPrintStream.print(lineSep +
iBase + "</" + name + ">");
    }
}
else
{
    outputPrintStream.print("/ >");
}

break;

case Node.TEXT_NODE:
case Node.CDATA_SECTION_NODE:

    String val = node.getNodeValue().trim();
    if (!whiteSpace(val))
    {
        // outputPrintStream.print( lineSep
+iBase+iInc+node.getNodeValue());

outputPrintStream.print(node.getNodeValue());
    }

    break;

case Node.PROCESSING_INSTRUCTION_NODE:
    outputPrintStream.print(lineSep + "<?" +
node.getNodeName() + " "
        + node.getNodeValue() + "?>");

    break;

```

```

        case Node.ENTITY_REFERENCE_NODE:
            outputPrintStream.print(lineSep + "&" +
node.getNodeName() + ";");

            break;

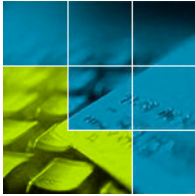
        case Node.DOCUMENT_TYPE_NODE:

            // The Document Type Node is NOT supported
            by the XML Input Component

            /* DocumentType docType =
(DocumentType)node;
            outputPrintStream.print(lineSep +
"<!DOCTYPE " + docType.getName());
            if (docType.getPublicId() != null) {
                outputPrintStream.print(" PUBLIC \"" +
                    docType.getPublicId() + "\" ");
            } else {
                outputPrintStream.print(" SYSTEM ");
            }
            outputPrintStream.print("\"" +
docType.getSystemId() + "\">");
            */
            break;
        }
    }

// Class XMLtoEngine

```



Chapter 4

XML Input Component Security

Security Protocols

The XML input component supports HTTP communication in both clear (`http://...`) and secure (`https://...`) modes. The secure hypertext transfer protocol (HTTPS) is a communication protocol designed to transfer encrypted information between computers over the World Wide Web. HTTPS is HTTP using a Secure Socket Layer (SSL). A secure socket layer is an encryption protocol invoked on a Web Server that uses HTTPS.

Secure Socket Layer (SSL) allows software to communicate with Web servers in a secure, encrypted manner. Many Web sites that conduct electronic commerce use SSL to securely transmit credit card numbers from a customer's Web browser to the Web server.

Generally speaking, a client using HTTPS with the XML input component does not need to do anything special to effect a successful transaction. However, the following *caveats* apply:

- The XML input component only supports 128-bit encryption strength ciphers. The following cipher suites are supported by the XML input component:
 - `SSL_RSA_WITH_RC4_128_SHA`
 - `SSL_RSA_WITH_RC4_128_MD5`
- It is assumed that the XML input component client is utilizing the appropriate support libraries to communicate using SSL. For example:
 - Web browsers: Microsoft and Netscape browser products support HTTPS communication. However, you may need to verify that your browser supports 128-bit encryption strength.
 - C/C++ clients: Third-party SSL software is generally required to communicate to a server. Examples of this would be OpenSSL and SSLPlus.

- Java clients: Third-party SSL software is also available for Java clients. One recommended library is from Sun, the Java Secure Socket Extensions (JSSE), which has a simple setup to support HTTPS.

Trusted Server Certificate

When the XML input component is configured, the appropriate trusted SSL server certificate should be obtained and installed on the machine in which the XML input component will be running. In a clustered ClearCommerce Engine environment, a trusted server certificate should be installed on all systems running the XML input component.

How To Get and Install a Secure Server Certificate

If the ClearCommerce Engine is intended to support external clients via HTTPS (using the XML input component), then you must obtain a 128-bit strength SSL server certificate. The following steps describe the procedure to obtain a trusted certificate:

- 1 cd to the appropriate directory containing the Engine certificate information (where *CCEbasedirectory* is the target directory for the ClearCommerce Engine):
 - NT: *CCEbasedirectory*\certs
 - Solaris/HP-UX: *CCEbasedirectory*/cfg
- 2 Create a config file that accurately reflects the server name for which you are getting a certificate. For a clustered Engine environment using a load balancer, create a config file for each server that reflects the common name shared by all of the systems in the cluster and by which the client will know the clustered environment):
 - NT: Copy the *cce.cnf* file that was created during Engine installation to another file

- Solaris/HP-UX: create a `cce.cnf` file in the `CCEbasedirectory/cfg` directory. It should look like the following (replace the {} items with the correct information):

```
[ req ]
distinguished_name = req_distinguished_name
prompt = no

[ req_distinguished_name ]
C = {two letter country code, e.g., US}
ST = {State, spelled out, e.g., Texas}
L = {City, e.g., Austin}
O = {Organization/Company name}
OU = {Organizational Unit, e.g., Sales (Make this
name unique for each certificate in a clustered
environment that share the same CN)}
CN = {correctly qualified server name/correctly
qualified cluster name}
Email = {email address}
```

Note: It is important to ensure that the CN entry in the config file accurately reflects your server name. (For a clustered environment, ensure the CN entry accurately reflects the common name used by all of the systems in the cluster and the name by which the client will know the clustered environment). For an internal test/development server, you can specify a server name only. For an Internet server, the fully qualified server name and domain work best. (There have been problems obtaining a test certificate from VeriSign when the IP address of the server is used).

- 3 For each system needing a trusted certificate, use the following command to create a private key (`server.key`) and Certificate Signing Request (CSR) in separate files (in this example, the config file(s) created in the previous step is named `server.cnf`):

```
../bin/openssl req -new -nodes -newkey rsa:1024 -
config server.cnf -out server.pem -keyout server.key
```

- 4 Send the contents of the `server.pem` file off to the Certificate Authority (VeriSign) that will provide the trusted SSL certificate.

- 5 When the SSL certificate is received from the CA, put it in a file in the directory mentioned previously. For example, *server.cer*.
- 6 Prepend the *server.key* file's content to the contents of *server.cer*. The resulting *server.cer* file should look similar to the following (note, the following is an example only, and is not valid):

```

-----BEGIN RSA PRIVATE KEY-----
MIICXQIBAAKBgQDWhjsGaFxmDLzDrmC2lv/KQANYESHGOyHDwZnK9NHskM+HiT3
+onyFbO/ jI6SzSbc9bBg23Ro3zP+9WE0S5R8aBhNYmR+virpxma5s7FY93+sTZtM
F3huqhvWor jKhhXsEKRKuhVmeSTtMr inNV/Gor IO5J0hh+LQ8IR14MUUuQIDAQAE
AoGBAIYfO07+ytFV0Pxut6GpVdofeyomy013PgdmUNNkbuTUCivBQ9SS975K+5g5
2L94ghPRQdWY/3QEghPq7CSORoTz66LxZb5JOXSnWWJSURZsU07Nw7Mw0CcXmaP1
lVfPlgEkPW8egRqf9UA+9breYJzqyUxo8yHaQ/dlAqi/tuHRAkEA+SvuoolEk7tZ
lrZN7EITHOcDy+nm8qf2geOhtNqB9uj+1wd2uGfDbctE6hTWJidIeTGP2VEJHncX
l+XCkAXRbQJBANwAWkCv9NwtG78rv7RrZTeBNVvUDSNYvIKX9sR821J+LUP/utFf
111ZC3+Duy5rKxpd061sPtg0ywgwrc2EDP0CQD0lxdtIXCre/r5YoMjQ8kXOf4UE
MrPr+jen+CAC23YPebKlV0IqRFtmZFQZpvAYBdYQDqA jHdzHFxgkv11+xAUCQFle
70r978uNq4rie9MQGAMNox51qwyj3Jh0oTM6wWHmjf7koQC3LVzncmtPmTeHI5Bc
4128716oOzfsEoS9UwUCQQCxFJF4n5mcrylzRy6wWw1me+837ZTmvZvtjfEPJBT8
G9GR6bc0MCs1Wczv2FUKDkd02ZRjqU9xR80dKzqVMiXt
-----END RSA PRIVATE KEY-----
-----BEGIN CERTIFICATE-----
MIICWTCCAgMCEHiYvNKsFWiEHpm8zzvDBtIwdQYJKoZlIhvcNAQEEBQAwgaxFjAU
BgNVBAoTDVZlcm1TaWduLCBjb250vcmVub3NpdG9yeS9UZnN0Q1BTIEluY29ycC4gQnkkgUmVmLiBmaWFiLiBMVEQuMUyV
RAYDVQQLLEZlG3IgwVvYyVnNpZ24gYXV0aG9yaXplZCB0ZXN0aW5nIG9ubHkuIE5v
IGFsc3VyYW5jZXMgKEMpVlMxOTk3MB4XDTAxMDQwMzAwMDAwMFOXDTAxMDQxNzIz
NTk1OVowcjELMAkGA1UEBhMCVVMxZjAMBgNVBAgTBVRleGFzMQ8wDQYDVQQHFAZE
dXN0aW4xIjAgBgNVBAoUUGUNsZWZyQ29tbWVyY2UgQ29ycG9yYXRpb24xDDAKBgNV
BASUA1BUUZEQMA4GA1UEAxQHCHJvZ2VycyCBnzANBgkqhkiG9w0BAQEFAAOBjQAw
gYkCgYEAdiIY7BmhcZnS8w65gtpb/ykADchEhxjshw8GZyvTR7JDPH4k9/qJ8hWz
v4yOks0mwvWwYnt0an8z/vVhNeuUfGgYTWJkfr4q6cZmubOxWpd/rE2bTBd4bqok
lqK4yoYV7BCkSroVZnkk7TK4pzVfxqKyDuSdIYfi0PCEZeDFFLkCAwEAATANBgkq
hkiG9w0BAQQFAANBAIPvq1vS1F2reK5iFG6mFyA6LRwiUSe/+ch3mRK0T1EsxTv9
rXoMOBwU4D7G/cyQNFqmK/wBpc7mqtkMLMsPvIE=
--END CERTIFICATE-----

```

This file makes up the trusted certificate file needed to use secure communications with the XML input component.

Using a Temporary Trusted Certificate

While awaiting the live certificate from the Certificate Authority (CA) of choice, it is possible, but not required, to obtain a temporary test SSL certificate from VeriSign.

However, to properly recognize the temporary certificate as a valid trusted certificate, you must obtain a “Test CA root certificate” from VeriSign and install it into your browser as a trusted certificate, or use the *keytool* command to install it as a trusted certificate in the client’s Java keystore.

Configuring the Java Client for a Temporary Certificate

Note: This procedure assumes that Java 1.3 or later is used.

- 1 cd to `java_home/jre/lib/security` directory. There should be a `cacerts` file in this directory.
- 2 Copy the existing `cacerts` file to a backup file.
- 3 Copy the file `getcacert.cer` from <http://www.Verisign.com/server/trial/faq/index.html>, right-click on the “Accept” button near the top of the page, and download the `getcacert.cer` file from that location.
- 4 Execute the following command:


```
keytool -import -trustcacerts -keystore cacerts -file
getcacert.cer
```
- 5 Type *changeit* to the following prompt:


```
Enter keystore password: changeit
```
- 6 Type *yes* to the following prompt:


```
Trust this certificate? [no]: yes
```

This will allow you to talk to a secure site that is using a temporary VeriSign SSL certificate.

To verify what certificates are included in the Java environment do the following:

- 1 cd to the `java_home/jre/lib/security` directory. There should be a `cacerts` file in this directory.
- 2 Enter the command:


```
keytool -list -v -keystore cacerts
```

Configuring a Web Browser for a Temporary Certificate

The same temp root certificate could be installed into a browser by, for example, pointing an Internet Explorer browser to the same VeriSign web page as described above, clicking the **left** button on the Accept graphic, and installing the certificate per the IE's certificate wizard.

Security Certificate Web Sites

- <http://www.thawte.com> - Secure certificate site
- <http://www.Verisign.com> - Secure certificate site
- <http://www.Verisign.com/rsc/wp/certshare/certshare.html> - information regarding trusted certificates for a clustered environment
- <http://webservices.web.cern.ch/WebServices/> - Information on secure certificates



Chapter 5

XML Input Component Configuration

You must perform the following tasks to fully configure the XML Input Component:

- Configure the XML Input Component table
- Configure the XML Input Component component

Important: In version 5.2, several configuration parameters were added, while others were deleted. If you used a previous version of the XML input component, you must update your configuration.

Configuring the XML Input Component Table

As of Version 5.2, the XML input component can be configured using the ClearCommerce Engine API. Previous version of the XML Input Component required manual configuration of the XML Input Component database table.

The installation of the ClearCommerce Engine (which includes the installation of the XML Input Component) sets up two ports for the XML Input Component to listen on. By default, port 11500 is for secure traffic,

while port 11501 is for clear traffic. These values can be changed by sending a ConfigDoc to the Engine. The following table lists the fields that can be used in the ConfigDoc.

Note: You must be granted permission to update the XML Input Row Configuration resource to configure the XML Input Component. See the *ClearCommerce Engine API Reference and Guide* for more information about roles and resources.

Table 5-1

Field	Default Value	Description
DirInst	0	The initial instance of the ClearCommerce Engine. This value must be incremented for each additional instance of the Engine installed for a clustered environment
Complnst	A	The component instance associated with the dir_inst above. This value does not change.
Port	11500 and 11501	The ports configured for the XML Input Component to listen on. Port 11500 is set up for secure communication, and 11501 is for clear communication. This field is configurable, and more ports can be added as necessary.
NewPort	Not applicable	The field is used to change port numbers. It can only be used with Update actions.
Encrypt	1 (for port 11500) & 0 (for port 11501)	This flag determines if the port is communicating using encryption (1) or clear mode (0). The default configuration sets up a port for each mode.

Note: Other parameters that were in present in versions prior to 5.2 have been deprecated.

The following example illustrates the structure of a document that changes port 11500 to 12500:

```

EngineDocList
  DocVersion = 1.0
  EngineDoc
    ContentType = ConfigDoc
    User
      Name = JDoe
      Password = dog
      ClientId = 1
    Instructions
      RoutingList
        Routing
          Name = CcxXmlInput
    ConfigDoc
      CompList
        Comp
          Name = CcxXmlInput
          ConfigActionList
            ConfigAction
              Name = XmlInputConfig
              Action = Update
              ValueList
                Value
                  DirInst = 0
                  CompInst = 0
                  Port 11500
                  NewPort 12500

```

See the *ClearCommerce Engine API Reference and Guide* for more information about using ConfigDocs to configure a component.

Configuring the XML Input Component Command Line Keys

The XML Input Component requires that several command line keys be configured on the Engine.

To configure the XML Input Component:

- 1 Log in to the System Administrator Tool. The Administration page is displayed.
- 2 Click **Engine** from the selections on the left side of the page. The Director Configuration page is displayed.
- 3 Click the **Instance ID** for the Engine instance with which you want to work.
In a non-clustered environment, the only Instance ID available is 0.
In a clustered environment, several Instance IDs might be available.
When you click the **Instance ID**, the Component Configuration page is displayed.
- 4 Click **Update** at the bottom of the page.
The page will refresh, and a new column titled **Select** will appear to the left of Instance ID.
There will be a radio button to the left of each component name.
- 5 Click the radio button to the left of the CcxXmlInput. Click **Update** at the bottom of this page.
- 6 A second Component Configuration page similar to the following is displayed.

Component Configuration	
Instance ID	11
Inst Name	A
Comp Name	CcxXmlInput
File Spec	CcxXmlInput
Thread Count	10
Q Depth	100

Commandline Parameters	
Key	Value
MaxNoRetries	3600
MaximumDocumentSize	0
MemQueueDepth	100
PendingQueueDepth	-1
ReadTimeout	0
RetryPeriod	1000
SslCertificate	

Figure 5-1 Component Configuration and Commandline Parameters for CcxSmlInput

Fields that can be changed are in a text box. In most cases, it is not necessary to change the Thread Count or Q Depth value.

- 7 In the Command-line Parameters section, enter parameters and values based the following information. If a command-line parameter is not displayed, enter the name of the parameter in the Key column.

Table 5-2

Command-line Parameter	Description	Value/Default
MaximumDocumentSize	Specifies the maximum size, in bytes, that a document sent to the Engine can be. Documents larger than this size will be rejected. The default value of 0 indicates there is no maximum size limit.	0
MaxNoRetries	The maximum number of retries the XML Listener makes in accepting connections on a port before it terminates.	3600
MemQueueDepth	Deprecated. Do not use this command-line parameter.	
PendingQueueDepth	The number of connections that will be queued by the TCP/IP communication layer that awaits completion of the TCP handshake. If a value of -1 is specified, the XML Input Component will use the queue depth defined on the machine running the ClearCommerce Engine	-1

Table 5-2 (Continued)

Command-line Parameter	Description	Value/Default
ReadTimeout	The maximum amount of time, in seconds, the XML Input Component will wait for data from the client on an established connection before timing out and aborting the TCP session. This does not affect further communication attempts.	0, indicating there is no limit on how long the XML Input Component will wait.
RetryPeriod	The time, in milliseconds, the XML Listener waits between attempts to accept client connections.	1000
SslCertificate	Fully qualified name of the certificate file to use for SSL. WARNING You must define this before you run transactions through the Engine.	Not applicable

Note: Do not edit existing key names. In cases where a default value for the key is set during installation of the ClearCommerce Engine, you will need to change only the value, not the key.

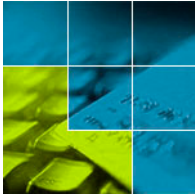
- 8 Click **Submit Modification** to save your changes or **Reset** to ignore them.
- 9 Change the component configuration and command line parameters for other components, as desired. When you are finished, log off from the System Administrator Tool. Then you must stop and start the Engine to load the changes.

The following sample document illustrates the structure of a document that changes the values of the command line parameters for the CcxXmlInput component:

```

EngineDocList
  DocVersion = 1.0
  EngineDoc
    ContentType = ConfigDoc
    User
      Name = JDoe           // User's name
      Password = dog        // User's password
      ClientId = 1          // Admin
    Instructions
      RoutingList
        Routing
          Name = Director
    ConfigDoc
      CompList
        Comp
          Name = Director // Required
        ConfigActionList
          ConfigAction
            Name = dir_comp_config // Required
            Action = Update
            ValueList
              Value
                CompName = CcxXmlInput
                MemQueueDepth = 150
                SslCertificate =
/usr/local/clearcommerce/cfg/CCE.pem

```

Chapter 6

XML Input Component Functionality

This chapter highlights some of the functionality associated with the XML input component that the client application needs to be aware of.

White Space Handling

The XML input component strips preceding and trailing “white space” characters (defined as spaces, tabs, and blank lines) from all of the ClearCommerce Engine datatype element values. The client can retain “white space” only for the **string** datatype. This can be done by specifying the following attribute in **ALL** of the string element tags that need “white space” preserved: **xml:space=“preserve”**. An example to preserve the “white space” in the ContentType value:

```
<ContentType xml:space="preserve">      OrderFormDoc
</ContentType>
```

XML Processing Instructions

If an incoming XML document contains processing instructions, they can only exist at the top of the document. An example of a processing instruction is highlighted in the following:

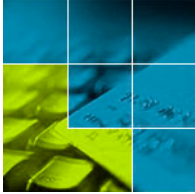
```
<?XML version="1.0" encoding="UTF-8"?>
<EngineDocList>
  <DocVersion>1.0</DocVersion>
  <EngineDoc>
    ...
  </EngineDoc>
</EngineDocList>
```

XML Encoding Declaration

The encoding declaration in the XML declaration line is optional. The XML input component accepts only “UTF-8” character encoding as input from a client. The “UTF-8” character representations must be within the Latin-1 (ISO-8859-1) character range. Based on the W3C XML Specification, “UTF-8” is the default encoding for XML documents. Therefore, if no encoding declaration is supplied in the XML declaration line then the XML input component assumes the encoding is UTF-8 and processes the document. However, if an encoding declaration is supplied with the incoming XML document, it must be “UTF-8” (case insensitive) or the XML input component returns an error to the client. The component converts the “UTF-8” character representations to “ISO-8859-1” for transactions passed to the ClearCommerce Engine. When the transaction processing is complete, the component converts the characters back to “UTF-8” representation to send back to the client.

An example of an XML encoding declaration line is highlighted in the following:

```
<?XML version="1.0" encoding="UTF-8"?>
```



Chapter 7

Troubleshooting

This chapter is divided into three sections:

- The Confidence Checks section describes the checks to perform to verify that the system is working correctly.
- The Errors section describes error conditions and what to do to fix them.
- The Corrective Actions section describes repair actions to correct problems in the system. Both the Confidence Checks and Errors sections refer to the Corrective Actions section.

Confidence Checks

It is beneficial to set up a series of confidence checks to assist in determining what might be wrong if a problem occurs. Each time an issue is encountered, go through the steps below to possibly help determine where the problem may be.

Perform the checks in the following order.

- 1** Is the ClearCommerce Engine up and running?
Use the User Interface (UI) to check that the ClearCommerce Engine is up and running. Enter the URL of your Engine in the browser and see if you can log in. If you can log in, then the Engine is running. Otherwise, restart the Engine or check the Web Server. Verify the ClientId, Username and Password that you are using in the XML document can access the UI. Also make sure they have the correct permissions.
- 2** Can a test transaction be sent from the UI?
Send a test transaction from the UI. Log in to a store, select the Order tab at top of the window, select Point of Sale, enter parameters and make sure to set Processing Mode to Y. If a response is not returned, then the ClearCommerce Engine is not properly functioning.

- 3 Can an XML document be sent to the XML input component from the browser HTML application? (See “Sample HTML Client” on page 16 for an example HTML application.)

Using the HTML tester created in Chapter 3, send a PreAuth transaction to the XML input component in *clear* mode. If a response is returned, but the transaction was not approved, then check the error message in the document to see if it provides any leads to the problem. Most of the time, the user/password/clientid combination is not correct. If the XML input component does not respond, then it is down or misconfigured. Try restarting the Engine. If this test works then try secure mode, if appropriate.

- 4 Can an XML document be sent to the XML input component from the Java code? (See “XMLtoEngineFromFile.java Example” on page 22 for an example.)

Using an application similar to the sample Java code, send an XML document to the Component. If you get a response then, but not approved, then the problem is in the Java code or document structure. The response messages from the Engine are very helpful. Again the document is normally misconfigured. Make sure you are not sending a secure transaction to a clear port. You can view the ClearCommerce Engine log to see if any error messages are being logged. On Solaris and HP-UX, the default location for the log is `/var/log/CcxEngine.log`. On Windows systems, the default location for the log is *CCEbasedirectory*.

- 5 Can a transaction be sent to the C or Java API?

To verify that the parameters that you are using and the ClearCommerce Engine can process transactions, use the C or Java API to execute a similar transaction you’re attempting in XML. If the same values work in the C or Java API, then the problem is in the Java code or the XML input component is not functioning properly.

- 6 Is the XML document valid?

If you suspect that the document is not in valid XML format, you can load it into an XML editor, like XMLSpy, which can validate the document is well-formed. The element data types should be verified against the data types of each field in the document hierarchy.

Errors

The majority of the problems experienced with the XML input component have been related to the XML document. Either the document is not well formed, invalid or the values are incorrect. Loading the document in an XML editor will help determine if it is valid and well-formed.

The following questions and answers might help:

- 1 Q:** I'm doing an integration and I'm getting a response that indicates that the "document format is invalid".

A: One likely problem is that the POSTed data that you are sending to the XML input component server is the straight XML. This data *must* be prefixed with the string **varname=** , where **varname** is whatever string you like. For example, "CLRCMRC_XML =" as a prefix would work just fine.
- 2 Q:** When I try to communicate with the XML input component server, my Java application gets an exception stating that the "https protocol was not found", or something of that nature.

A: This might mean that you are trying to use the Java Secure Sockets Extension (JSSE) and have not properly set up the environment such that the Security manager can find the JSSE HTTPS protocol handler. Reference the *JSSE Guide* for help in this area. (Remember that system properties can be set via the **-D** parameter to the **java** command, or via **System.setProperty(...)** in the code).
- 3 Q:** When I run my Java sample code, I get a "Connect refused:" error.

A: Check to see if there is a reference in the error to SSLSocketImpl. If there is, then your certificate might have expired.
- 4 Q:** I get a response back from the Engine that has MaxSev="6".

A: You got an error from the ClearCommerce Engine. Check the error message "Text" field for details about the error. Transaction errors are covered in detail in the *ClearCommerce Engine API Reference and Guide*.
- 5 Q:** I'm sending a perfectly valid XML request to the XML IC, but getting the following error message: "The XML document encoding declaration is invalid."?

A: If you are sending an XML document that contains an encoding declaration it must be "UTF-8". Otherwise, if no encoding declaration is specified, the XML IC defaults to "UTF-8".
- 6 Q:** I'm sending valid "UTF-8" character representations to the XML IC but getting the following error message: "The UTF-8 character representation is not in the Latin-1 character range."?

A: The “UTF-8” character representations must be within the Latin-1 (ISO-8859-1) character range. The ClearCommerce Engine supports the Latin-1 character set.

7 Q: I'm sending what seems to be a valid XML document to the XML IC but I am getting the following error (or something similar) "Value for element 'BirthDate' is not valid. ('argument' is 'strStartDate', 'type' is 'CcaString01', 'value' is ")?"

A: Any non-string data element that contains an empty value will be rejected by the ClearCommerce Engine and produce the above error (or one similar based on the element name). The fix for this is to strip empty data elements from the XML document before sending it to the XML IC.

8 Q: I am sending what seems to be a valid XML document to the XML IC but it contains an empty Periodic Billing Order record (<PbOrder>) and I get the following error "Unable to cast object from 'CcaString01_t' to 'CcaVariantMap01_t' in context 'CcaVariant01::OperatorType'. ('path' is 'EngineDoc.OrderFormDoc.PbOrder')?"

A: If you are not planning to use the Periodic Billing record in the XML document, do not include the <PbOrder> element tag as part of the document sent to the XML IC.

Corrective Actions

The following actions can be performed to help resolve problems:

- Restart the Engine.

See the *System Administrator Guide* for this procedure.

- Using a database tool, like SQLPlus, check to see how the XML Input Component is configured. Here are the most common SQL commands to use:

To check the XML Input Component configuration table, especially the port, encryption and thread count.

```
select * from xml_input_config;
```

To check the Component configuration table for the XML Input Component SSL certificate location:

```
select * from dir_comp_config;
```

- Send in the correct URL (HTTPS | HTTP, Engine, port).

The URL that you are sending to the Engine is not correct. Make sure you can ping the ClearCommerce Engine machine. Next, verify that you are using the appropriate protocol, either HTTPS vs. HTTP. Then, verify that you are using the port that is compatible with the protocol. By default, HTTPS requests go to port 11500, and HTTP requests go to port 11501.

- Correct the values of the **User**, **Password** and **ClientId** fields.

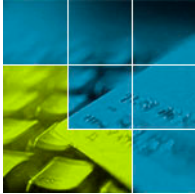
Make sure that the values for the **User**, **Password**, and **ClientId** fields are correct for the entity. The best way to check this is to use the UI to log in as this user. For example, assuming that you are using the following XML code:

```
<User>
  <Name>XML_store</Name>
  <Password>xml</Password>8
  <ClientId DataType="S32">20</ClientId>
</User>
```

You can access the Engine via the UI by entering an URL in a browser like this:

```
https://<CCEhost>/cgi-bin/ClearCommerce_Engine/20
```

When prompted enter the Name and Password, such as `XML_store` and `xml` in the example above.



Index

C

- clustering
 - and component configuration 60
- Component Configuration Database Table 59
- components
 - configuring 60
- Confidence Checks 67
- ConfigDocs
 - dir_comp_config 63
 - dir_config 59
- configuring
 - components 60
- Configuring a web browser for a temporary certificate 56
- Configuring the Java Client for a temporary certificate 55
- Corrective Actions 70
- Create an XML document with JAXP and send to ClearCommerce Engine 26

D

- Developer Support (Web) Site xiii, 5
- Document Object Model (DOM) 3
- DTD 5

E

- Errors - Q & A 69

H

- How To Get and Install a Secure Server Certificate 52
- HTTP POST 4

I

- Introduction 1

J

- Java and XML 1
- Java API for XML Parsing (JAXP) 3
- Java Secure Socket Extension (JSSE) 4

K

- key/value pair 62

P

- PendingQueueDepth (field) 61

R

- Read XML document from a file and send to ClearCommerce Engine 20

S

- Sample Client Applications 15
- Sample HTML Client 16
- Sample Java Clients 20
- Sample XML Document 11
- Security Certificate Websites 56
- Security Protocols 51
- Simple API for XML (SAX) 2
- SslCertificate (field) 62

T

- Troubleshooting 67
- Trusted Server Certificate 52

U

Using a Temporary Trusted Certificate 54

V

valid 5

validating your XML 5

W

well-formed 5

What is XML 1

White Space Handling 65

X

XML and ClearCommerce 4

XML and the ClearCommerce API

Document Hierarchy 8

XML Encoding Declaration 66

XML Input Component Configuration 57

XML Input Component Database Table

57

XML Input Component Functionality 65

XML Input Component Security 51

XML Input to the ClearCommerce Engine

7

XML Input to the Engine 8

XML Interface 7

XML Processing Instructions 65

XML Related Tools 6

XML Websites 5

XML Websites and Tools 8