CyberSource Simple Order API Client

Developer Guide

September 2015

CyberSource®
the power of payment

CyberSource Corporation HQ | P.O. Box 8999 | San Francisco, CA 94128-8999 | Phone: 800-530-9095

CyberSource Contact Information

For general information about our company, products, and services, go to http://www.cybersource.com.

For sales questions about any CyberSource Service, email sales@cybersource.com or call 650-432-7350 or 888-330-2300 (toll free in the United States).

For support information about any CyberSource Service, visit the Support Center at http://www.cybersource.com/support.

Copyright

Restricted Rights Legends

For Government or defense agencies. Use, duplication, or disclosure by the Government or defense agencies is subject to restrictions as set forth the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 and in similar clauses in the FAR and NASA FAR Supplement.

For civilian agencies. Use, reproduction, or disclosure is subject to restrictions set forth in subparagraphs (a) through (d) of the Commercial Computer Software Restricted Rights clause at 52.227-19 and the limitations set forth in CyberSource Corporation's standard commercial agreement for this software. Unpublished rights reserved under the copyright laws of the United States.

Trademarks

CyberSource, The Power of Payment, CyberSource Payment Manager, CyberSource Risk Manager, CyberSource Decision Manager, CyberSource Connect, Authorize.Net, and eCheck.net are trademarks and/or service marks of CyberSource Corporation. All other brands and product names are trademarks or registered trademarks of their respective owners.

Contents

Recent Revisions to This Document

Release	Changes
September 2015	▪ Updated the production server URL and the test server URL.
September 2014	▪ Added chapter to document the new .NET 4.0 client. See ".NET 4.0 Client," page 216.
April 2013	▪ Noted that all of the Simple Order API clients except the .NET 4.0 client are supported only on 32-bit operating systems. ▪ Combined all Simple Order API client documents into this developer guide, which covers all supported programming languages.
January 2013	▪ Noted a change in transaction security key use requirements for the CyberSource production and test environments. For more information, see the “Transaction Security Keys” section in each programming language chapter: • ASP client • C/C++ client • .NET 1.1 client • .NET 2.0 client • Java client • Perl client • PHP client ▪ Removed information about how to generate security keys and added it to *Creating and Using Security Keys* (PDF | HTML).

About This Guide

Audience

This guide is written for application developers who want to use the CyberSource Simple Order API client to integrate the following CyberSource services into their order management system:

- CyberSource Essentials
- CyberSource Advanced

Using the Simple Order API client SDK requires programming skills in one of the following programming languages:

- ASP/COM
- C, C++
- Java/Cold Fusion
- .NET
- Perl
- PHP

To use these SDKs, you must write code that uses the API request and reply fields to integrate CyberSource services into your existing order management system.

Purpose

This guide describes tasks you must complete to install, test, and use the CyberSource Simple Order API client software.

Scope

This guide describes how to install, test, and use all available Simple Order API clients. It does not describe how to implement CyberSource services with the Simple Order API. For information about how to use the API to implement CyberSource services, see "Related Documents," page 21.

Conventions

Note, Important, and Warning Statements

Note

A *Note* contains helpful suggestions or references to material not contained in this document.

Important

An *Important* statement contains information essential to successfully completing a task or learning a concept.

Warning

A *Warning* contains information or instructions, which, if not heeded, can result in a security risk, irreversible loss of data, or significant cost in time or revenue or both.

Text and Command Conventions

Convention	Usage
bold	■ Field and service names; for example: Include the **ics_applications** field. ■ Items that you are instructed to act upon; for example: Click **Save**.
italic	■ Filenames and pathnames. For example: Add the filter definition and mapping to your *web.xml* file. ■ Placeholder variables for which you supply particular values.

Convention	Usage
`monospace`	■ XML elements. ■ Code examples and samples. ■ Text that you enter in an API environment; for example: Set the **davService_run** field to `true`.

Important The Simple Order API was originally referred to as the *Web Services API* in CyberSource documentation. References to the Web Services API may still appear in some locations.

Related Documents

Client Package Documentation

The following documentation is available in the client package download:

- `README` file
- `CHANGES` file
- Sample code files

CyberSource Services Documentation

This guide (Simple Order API Client Developer Guide) contains information about how to:

- Create the request
- Send the request
- Receive the reply

In contrast, CyberSource services documentation listed in Table 1 contains information about how to:

- Determine what to put in requests sent to CyberSource.
- Interpret what is contained in the reply from CyberSource.

Each type of CyberSource service has associated documentation:

Table 1 CyberSource Services Documentation

Type of Service	Available Documentation
CyberSource Essentials	■ *Credit Card Services User Guide* (PDF | HTML) ■ *Electronic Check Services User Guide* (PDF | HTML)
CyberSource Advanced	■ *Credit Card Services Using the Simple Order API* (PDF | HTML) ■ *Reporting Developer Guide* (PDF | HTML)

If you use other CyberSource services, the documentation can be found on the CyberSource Essentials or CyberSource Advanced (Global Payment Services) sections of the CyberSource web site.

Customer Support

For support information about any CyberSource service, visit the Support Center at:

http://www.cybersource.com/support

CHAPTER 1

Introduction

Important: Only the .NET 4.0 client for the Simple Order API is supported on both 32-bit and 64-bit operating systems. All of the other Simple Order API clients are supported on 32-bit operating systems only.

The CyberSource Simple Order API enables you to access CyberSource services using name-value pairs, XML, or the Simple Object Access Protocol (SOAP). The Simple Order API SDKs provide the client software for the following programming languages:

- ASP
- C, C++
- .NET version 1.1 and version 2.0
- Java
- Perl
- PHP

The Simple Order API is a good choice for businesses who:

- Must access CyberSource services that can only be accessed with APIs
- Have high volumes of transactions that warrant high levels of automation
- Must control and customize their customers’ buying experience
- Have an order page that is secured with Secure Sockets Layer (SSL)
- Can provide skilled software programmers to implement CyberSource services with the API

ASP Client

CHAPTER 2

Important

- The ASP client for the Simple Order API is supported on 32-bit operating systems only.
- If you are building an application to sell to others, see Appendix A, "Using the Client Application Fields," on page 390. This appendix has a list of API fields you can use in your request that describe the application, its version, and its user. If you use these fields in your request, you can view their values in the Transaction Search Details window of the Business Center.

Using ASP in a Hosted Environment

If you are operating in a hosted environment (with an Internet Service Provider hosting your web store), then read this section.

To use the CyberSource Simple Order API client for ASP, you must register several Microsoft Dynamic-Link Libraries DLLs. These DLLs ensure that your transactions are secure while being sent to CyberSource. If you use a hosted environment, you must check with your hosting provider (ISP) to make sure that they support the registration of custom DLLs. If you are unable to find any documentation related to your hosting provider's support of DLLs, then contact them with the following statement:

> **CyberSource requires the registration of three DLLs for use by my e-commerce software. These DLLs must be registered using regsvr32. CyberSource ensures the safety and functionality of these DLLs. Please let me know your policy for supporting this implementation.**

Note that it is also possible that other merchants who use your hosting provider may also use CyberSource, and so the hosting provider may have already installed the CyberSource ASP client and registered the DLLs. In that case, CyberSource recommends you verify with your hosting provider which version of the client they have installed and registered. If the client you want to use is newer, ask them to register the newer DLLs.

If you have any questions regarding the above information or installation of the client, please contact Customer Support.

Choosing Your API and Client

API Variation

With this client package, you can use either of these variations of the Simple Order API:

- Name-value pairs, which are simpler to use than XML
- XML, which requires you to create and parse XML documents

The test that you run immediately after installing the client uses name-value pairs.

Client Versions

CyberSource updates the Simple Order API on a regular basis to introduce new API fields and functionality. To identify the latest version of the server-side API for CyberSource services, go to https://ics2wsa.ic3.com/commerce/1.x/transactionProcessor.

The Simple Order API Client for ASP also has a version, but it is not the same as the API version. The client version represents the version of the client-side code that you use to access CyberSource services.

When configuring the client, you indicate which version of the API you want to use. When setting this parameter, do not use the current version of the client; use the current version of the API.

Sample Code

The client contains sample scripts and sample ASP pages that you can use to test the client.

Basic ASP Page Example

The example below shows the primary VBScript code required to send a Simple Order API request for credit card authorization and process the reply. The example uses name-value pairs. For a more complete example, see the sample program and sample ASP pages included in the package (see "Sample Scripts," page 27). "Using Name-Value Pairs," page 48 shows you how to create the code.

```
' Set oMerchantConfig properties
Dim oMerchantConfig
set oMerchantConfig =
    Server.CreateObject( "CyberSourceWS.MerchantConfig" )
oMerchantConfig.MerchantID = "infodev"
oMerchantConfig.KeysDirectory = "\keys"
oMerchantConfig.SendToProduction = "0"
oMerchantConfig.TargetAPIVersion = "1.18"

' Create request hashtable
Dim oRequest
set oRequest = Server.CreateObject( "CyberSourceWS.Hashtable" )

' We want to do credit card authorization in this example
oRequest.Value( "ccAuthService_run" ) = "true"

' Add required fields
oRequest.Value( "billTo_firstName" ) = "Jane"
oRequest.Value( "billTo_lastName" ) = "Smith"
oRequest.Value( "billTo_street1" ) = "1295 Charleston Road"
oRequest.Value( "billTo_city" ) = "Mountain View"
oRequest.Value( "billTo_state" ) = "CA"
oRequest.Value( "billTo_postalCode" ) = "94043"
oRequest.Value( "billTo_country" ) = "US"
oRequest.Value( "billTo_email" ) = "jsmith@example.com"
oRequest.Value( "card_accountNumber" ) = "4111111111111111"
oRequest.Value( "card_expirationMonth" ) = "12"
oRequest.Value( "card_expirationYear" ) = "2010"
oRequest.Value( "purchaseTotals_currency" ) = "USD"

' There are two items in this example
oRequest.Value( "item_0_unitPrice" ) = "12.34"
oRequest.Value( "item_1_unitPrice" ) = "56.78"
```

```
' Add optional fields here according to your business needs

' create Client object and send request
dim oClient
set oClient = WScript.CreateObject( "CyberSourceWS.Client" )
dim varReply, nStatus, strErrorInfo
nStatus = oClient.RunTransaction( _
          oMerchantConfig, Nothing, Nothing, oRequest, _
          varReply, strErrorInfo )

' Handle the reply. See "Handling the Return Status," page 50.
ProcessReply nStatus, varReply, strErrorInfo
```

Sample Scripts

The client contains two sample scripts, one for using name-value pairs and one for using XML. See "Testing the Client," page 30 or see the README file for more information about using the `AuthCaptureSample.wsf` script to test the client.

- For name-value pairs: See `AuthCaptureSample.wsf` in *<installation directory>*`\samples\nvp`.
- For XML: CyberSource recommends that you examine the name-value pair sample code listed above before implementing your code to process XML requests.

For the XML sample code, see `AuthSample.wsf` in *<installation directory>*`\samples\xml`. Also see the `auth.xml` XML document that the script uses.

Sample ASP Pages

The client download package also includes sample ASP pages in the *<installation directory>*`\samples\store` directory. You also have shortcuts to several of the files from the Start menu at **Start** > **Programs** > **CyberSource Simple Order API for ASP** > **Sample Store**.

Table 2 Files in aspSample Directory

File	Description
`global.asa`	Sets up the MerchantConfig object and stores it in a Session variable.
`EditFile.bat`	Batch file used with the Start menu program shortcuts. Not used directly with the ASP sample pages.
`checkout.asp`	Displays the contents of the shopping basket, prompts for address and payment information.
`checkout2.asp`	Authorizes the order and displays the result.
`store_footer.asp`	Footer used in the checkout pages.

Table 2 Files in aspSample Directory (Continued)

File	Description
`store_header.asp`	Header used in the checkout pages.

To use the sample ASP pages:

Step 1 In Microsoft Internet Information Services (IIS), create a virtual directory that points to the *`<installation directory>`*`\ samples\store` directory.

Step 2 Go to **Start** > **Programs** > **CyberSource Simple Order API for ASP** > **Sample Store** > **Edit configuration script (global.asa)** and set the properties of the MerchantConfig object to the correct values. For more information about the MerchantConfig settings, see "MerchantConfig Properties," page 34.

Step 3 If you have enabled logging, make sure that the Internet guest user account (default is IUSR_<*machine name*>) has write permission on the `logs` directory.

Step 4 Open a web browser and type the following URL:

```
http://localhost/<virtual directory>/checkout.asp
```

Installing and Testing the Client

Minimum System Requirements

- Windows Installer 2.0 or later

  If the Microsoft installer file (.msi) does not run, you may not have a suitable Windows Installer. You can download version 2.0 or a later version from the Microsoft Windows Installer Downloads page.

- Windows 2000 or XP, or later
- If using Windows XP: MSXML 4.0 Service Pack 2 (Microsoft XML Core Services)
- If using an operating system earlier than Windows 2000 SP3: WinHTTP 5.0

Note CyberSource recommends that your system use at least Windows 2000 SP3 because it comes with WinHTTP 5.1, which fixes bugs in the 5.0 version.

The SDK supports UTF-8 encoding.

Important Failure to configure your client API host to a unique, public IP address will cause inconsistent transaction results.

The client API request ID algorithm uses a combination of IP address and system time, along with other values. In some architectures this combination might not yield unique identifiers.

Transaction Security Keys

The first thing you must do is create your security keys. The client uses the security key to add a digital signature to every request that you send. This signature helps ensure that no one else can use your CyberSource account to process orders. You specify the location of your key when you configure the client.

Important You must generate two transaction security keys—one for the CyberSource production environment and one for the test environment. For information about generating and using security keys, see *Creating and Using Security Keys* (PDF | HTML).

Warning You must protect your security key to ensure that your CyberSource account is not compromised.

Installing the Client

To install the client:

Step 1 Go to the client downloads page on the CyberSource web site.

Step 2 Download the latest client package. You can save the file in any directory.

Step 3 Double-click the downloaded msi file.

The installation wizard opens.

Step 4 Follow the instructions on the screen.

The client is installed on your system. The default installation directory for the package contents is the `c:\simapi-asp-n.n.n` directory. The Start menu includes a new item for the CyberSource Simple Order API for ASP.

Step 5 Test the client. See "Testing the Client," page 30.

Testing the Client

After you install the client, test it immediately to ensure that the installation is successful.

To test the client:

Step 1 Modify the test script to include your merchant ID and the directory to your test environment security key:

a Choose **Start > Programs > CyberSource Simple Order API for ASP > NVP Sample > Edit configuration script (config.vbs)**.

b In the `config.vbs` script, replace the default values for your merchant ID and the directory in which your test environment security key is stored. The lines of code look like this before you edit them:

```
oMerchantConfig.MerchantID = "your_case_sensitive_merchant_id"

oMerchantConfig.KeysDirectory = "your_keys_dir(e.g.
baseDir\simapi-net-n.n.n\keys)"
```

c Save the script.

Step 2 Run the test by choosing **Start > Programs > CyberSource Simple Order API for ASP > NVP Sample > Run test script (AuthCaptureSample.wsf)**. This test requests an authorization and then a follow-on capture if the authorization is successful.

A command shell opens and displays the results of the test.

- If the test is successful, a decision of ACCEPT appears for both the credit card authorization and the follow-on capture.
- If the test is not successful, a different decision value or an error message appears.

To troubleshoot if the test fails:

Step 1 Verify that the `config.vbs` changes that you made in Step b above are correct.

Step 2 Run the test again.

Step 3 If the test still fails, look at the error message and find the return status value (a number from 1 to 8).

Step 4 See the descriptions of the status values in "Possible Return Status Values," page 45, and follow any instructions given there for the error you received.

Step 5 Run the test again.

Step 6 If the test still fails, contact Customer Support.

The client is installed and tested. You are ready to create your own code for requesting CyberSource services. For information about creating requests, see "Using Name-Value Pairs," page 48 if you plan to use name-value pairs, or "Using XML," page 58 if you plan to use XML.

Going Live

When you complete all of your system testing and are ready to accept real transactions from your customers, your deployment is ready to *go live*.

CyberSource Essentials Merchants

If you use CyberSource Essentials services, you can use the Business Center site to go live. For a description of the process of going live, see the “Steps for Getting Started” section in *Getting Started with CyberSource Essentials*.

Important You must also configure your client so that it sends transactions to the production server and not the test server. See the description of the MerchantConfig property "SendToProduction," page 34.

After your deployment goes live, use real card numbers and other data to test every card type you support. Because these are real transactions in which you are buying from yourself, use small monetary amounts to do the tests. Process an authorization, then capture the authorization, and later refund the money. Use your bank statements to verify that money is deposited into and withdrawn from your merchant bank account as expected. If you have more than one CyberSource merchant ID, test each one separately.

CyberSource Advanced Merchants

If you use CyberSource Advanced services, see the “Steps for Getting Started” chapter in *Getting Started with CyberSource Advanced* for information about going live.

When your deployment goes live, your CyberSource account is updated so that you can send transactions to the CyberSource production server. If you have not already done so, you must provide your banking information to CyberSource so that your processor can deposit funds to your merchant bank account.

After CyberSource confirms that your deployment is live, make sure that you update your system so that it can send requests to the production server (`ics2wsa.ic3.com`) using your security keys for the production environment. The test server (`ics2wstesta.ic3.com`) cannot be used for real transactions. For more information about sending transactions to the production server, see the description of the MerchantConfig property "SendToProduction," page 34.

After your deployment goes live, use real card numbers and other data to test every card type, currency, and CyberSource application that your integration supports. Because these are real transactions in which you are buying from yourself, use small monetary amounts to do the tests. Use your bank statements to verify that money is deposited into and withdrawn from your merchant bank account as expected. If you have more than one CyberSource merchant ID, test each one separately.

Deploying the Client to Another Computer

You can deploy the client to another computer without running the installer that CyberSource provided.

To deploy the client to another computer:

Step 1 On the destination computer, copy all the binaries (`*.dll` and `*.pdb`) into the *`<installation directory>`*`/lib` directory, or include them in your own installer.

Step 2 If the destination computer does not have MSXML 4.0 Service Pack 2 (Microsoft XML Core Services), copy the following files from the `%WINDIR%\system32` directory into the same directory on the destination computer:

`msxml4.dll`

`msxml4r.dll`

Step 3 Register `msxml4.dll`:

a Open a command prompt.

b Go to the `%WINDIR%\system32` directory.

c At the prompt, type `regsvr32 msxml4.dll`

Step 4 Register the CyberSource `*.dll` files:

a Open a command prompt.

b Go to the directory in which the binaries are installed on the destination computer.

c At the prompt, type

```
regsvr32 CybsWSSecurity.dll
regsvr32 CyberSourceWS.dll
```

You have deployed the client to the destination computer.

Updating the Client to Use a Later API Version

CyberSource periodically updates the Simple Order API (previously called the Web Services API). You can update your existing client to work with the new API version. Go to https://ics2wsa.ic3.com/commerce/1.x/transactionProcessor/ for a list of the available API versions.

To update the client to use a later API version, update the value for the `TargetAPIVersion` in the MerchantConfig object (see Table 3, "MerchantConfig Properties," on page 34). For example, to use the 1.18 version of the API, set the `TargetAPIVersion` to `"1.18"`.

Client Objects

MerchantConfig

The MerchantConfig object controls these types of settings for the client:

- Security
- Server
- Timeout
- Logging

MerchantConfig Properties

The data type for all of the properties is string.

Table 3 MerchantConfig Properties

Property	Description
LogString	Returns a string containing a comma-separated list of the MerchantConfig object's properties and their values.
MerchantID	Merchant ID. If you specify a merchant ID in the request (either using the Hashtable object if using name-value pairs, or the XML document if using XML), the one in the request takes precedence over the one in the MerchantConfig object.
KeysDirectory	Location of the merchant's security key. UNC paths are allowed. The client includes a `keys` directory that you can use. Make sure that you use your test security key for the test environment and that you use your production security key for the production environment. **Note** CyberSource recommends that you store your key locally for faster request processing.
SendToProduction	Flag that indicates whether the transactions for this merchant should be sent to the production server. Use one of these values: - `"0"`: Do not send to the production server; send to the test server. - `"1"`: Send to the production server.
TargetAPIVersion	Version of the Simple Order API to use, such as `1.18`. Do not set this property to the current version of the client. Instead, set it to an available API version. See "Client Versions," page 25 for more information. **Note** Go to https://ics2wsa.ic3.com/commerce/1.x/transactionProcessor/ for a current list of the available versions. See the *Simple Order API Release Notes* for information about what has changed in each version.

Table 3 MerchantConfig Properties (Continued)

Property	Description
KeyFilename	Name of the security key filename for the merchant, for example `<`***`keyName`***`>.p12`.
ServerURL	Alternative server URL to use. See "Using Alternate Server Properties" below for more information. Give the complete URL because it will be used exactly as you specify here.
NamespaceURI	Alternative namespace URI to use. See "Using Alternate Server Properties" below for more information. Give the complete namespace URI because it will be used exactly as you specify here.
EffectiveKeyFilename	If AlternateKeyFilename is not empty, this property simply returns the value of AlternateKeyFilename. If AlternateKeyFilename is empty, the property looks for a bstrMerchantID parameter that you provide. If you do not provide one, it derives the key filename from the MerchantID property.
EffectiveServerURL	If AlternateServerURL is not empty, this property simply returns the value of AlternateServerURL. Otherwise, it derives the server URL from the TargetAPIVersion and SendToProduction properties.
EffectiveNamespaceURI	If AlternateNamespaceURI is not empty, this property simply returns the value of AlternateNamespaceURI. Otherwise, it derives the namespace URI from the property TargetAPIVersion. This is particularly useful if you are passing in the request as XML and do not want to hardcode the namespace URI in the XML string. You can call this property to dynamically set the namespace URI of your `<requestMessage>` element.
Timeout	Length of timeout in seconds. The default is 110.

Table 3 MerchantConfig Properties (Continued)

Property	Description
EnableLog	Flag directing the client to log transactions and errors. Possible values: - `0`: Do not enable logging. - `1`: Enable logging. This property is ignored if you use a "Logger" object with the request. PCI **Important** Logging can cause very large log files to accumulate. Therefore, CyberSource recommends that you use logging only when troubleshooting problems. To comply with all Payment Card Industry (PCI) and Payment Application (PA) Data Security Standards regarding the storage of credit card and card verification number data, the logs that are generated contain only masked credit card and card verification number data (CVV, CVC2, CVV2, CID, CVN). Follow these guidelines: - Use debugging temporarily for diagnostic purposes only. - If possible, use debugging only with test credit card numbers. - Never store clear text card verification numbers. - Delete the log files as soon as you no longer need them. - Never send email to CyberSource containing personal and account information, such as customers' names, addresses, card or check account numbers, and card verification numbers. For more information about PCI and PABP requirements, see www.visa.com/cisp.
LogDirectory	Directory to which to write the log file. UNC paths are allowed. Note that the client will not create this directory for you; you must specify an existing directory.The client includes a `logs` directory that you can use. This property is ignored if you use a "Logger" object with the request.
LogFilename	Log filename. The client uses `cybs.log` by default. This property is ignored if you use a "Logger" object with the request.
LogMaximumSize	Maximum size in megabytes for the log file. When the log file reaches this size, it is archived into `cybs.log.`<***yyyymmdd***`T`***hhmmssxxx***> and a new log file is started. The ***xxx*** indicates milliseconds. The default value is `"10"`. This property is ignored if you use a "Logger" object with the request.

Example Setting Merchant Configuration Object Properties

This example sets the MerchantID property of a MerchantConfig object.

```
Dim oMerchantConfig

set oMerchantConfig =
    Server.CreateObject( "CyberSourceWS.MerchantConfig" )

oMerchantConfig.MerchantID = "merchant123"
```

Using Alternate Server Properties

Use the ServerURL and NamespaceURI properties if CyberSource changes the convention used to specify the server URL and namespace URI, but has not updated the client yet.

For example, these are the server URLs and namespace URI for accessing the CyberSource services using the Simple Order API version 1.18:

- Test server URL:

  `https://ics2wstesta.ic3.com/commerce/1.x/transactionProcessor`

- Production server URL:

  `https://ics2wsa.ic3.com/commerce/1.x/transactionProcessor`

- Namespace URI:

  `urn:schemas-cybersource-com:transaction-data-1.18`.

If in the future CyberSource changes these conventions, but does not provide a new version of the client, you can configure your existing client to use the new server and namespace conventions required by the CyberSource server.

MerchantConfig Method

MerchantConfig has one method:

Copy

Table 4 Copy Method

Syntax	`Copy( oMerchantConfig, fNonEmptyOnly )`
Description	Copies the properties of `oMerchantConfig`.
Returns	Nothing
Parameters	■ `oMerchantConfig`: The specific MerchantConfig object. ■ `fNonEmptyOnly`: Flag indicating which properties to copy. Set to `true` to copy only the properties that are not null or empty. Set to `false` to copy all properties, whether or not null or empty.

ProxyConfig

The ProxyConfig object controls proxy settings.

ProxyConfig Properties

The data type for all of the properties is string.

Table 5 ProxyConfig Properties

Property	Description
LogString	Returns a string containing a comma-separated list of the ProxyConfig object's properties and their values.
ProxySetting	Type of proxy server to use. Use one of these values: ■ `"0"`: Use WinHTTP proxy configuration, which is set using `proxycfg.exe`, a WinHTTP tool available in Windows 2000 SP3 and later, or as part of the Resource Kit in earlier Windows versions. ■ `"1"`: Do not use a proxy server. ■ `"2"`: Use the proxy server specified in the ProxyServer property.
ProxyServer	Proxy server to use. Only used if ProxySetting=`"2"`. Allowable formats include: ■ *`<http://>server<:port>`* ■ *`<http://>IP address<:port>`* The `http://` and `port` are optional.
Username	User name used to authenticate against the proxy server if required. If the proxy server requires the domain name during authentication, add the domain name and a backslash: ***`<domain>`***`\`***`<username>`***
Password	Password used to authenticate against the proxy server, if required.

This following example sets a ProxyConfig object to use a proxy server.

Example Setting ProxyConfig Properties

```
Dim oProxyConfig

set oProxyConfig =
    Server.CreateObject( "CyberSourceWS.ProxyConfig" )

oProxyConfig.ProxySetting = "2"

oProxyConfig.ProxyServer = varServer ‘ variable that contains server

oProxyConfig.Username = varUsername  ‘ variable that contains

                                     ‘ username

oProxyConfig.Password = varPassword  ‘ variable that contains

                                     ‘ password
```

ProxyConfig Method

ProxyConfig has one method:

Copy

Table 6 Copy Method

Syntax	`Copy( oProxyConfig, fNonEmptyOnly )`
Description	Copies the properties of `oProxyConfig`.
Returns	Nothing
Parameters	▪ `oProxyConfig`: The specific ProxyConfig object. ▪ `fNonEmptyOnly`: Flag indicating which properties to copy. Set to `true` to copy only the properties that are not null or empty. Set to `false` to copy all properties, whether or not null or empty.

Hashtable

You use the Hashtable object to store the request and reply information. You use the Hashtable object only if you are using name-value pairs, and not XML.

Hashtable Properties

The data type for all of the properties is string.

Table 7 Hashtable Properties

Property	Description
LogString	Returns a newline-character-separated list of the Hashtable object’s properties and their values.
Value	Reads or writes a value for the specified name. Accepts the name as a parameter. See the example below.
Content	Reads the content of the Hashtable as a string. Accepts a delimiter as a parameter, which is used to concatenate the name-value pairs. See the example below.
Names	Returns the array of name-value pairs. See the example below.
Overwrite	Gets or sets the overwrite mode. Allowable values: ▪ `True`: If the name that is being set already exists, its value is replaced with the new one. This value is the default value upon instantiation. ▪ `False`: The current value for the name is kept. The overwrite mode that you set is used for all field assignments until you reset it to the opposite value. See the example below.

The following example shows how to set name-value pairs in a Hashtable object:

Example Using the Value Property of the Hashtable Object

```
Dim oRequest
set oRequest = Server.CreateObject( "CyberSourceWS.Hashtable" )
oRequest.Value( "ccAuthService_run" ) = "true"
oRequest.Value( "card_accountNumber" ) = "4111111111111111"
```

Note

For the Hashtable object, the Value property is the default property. Thus you can leave out the “`.Value`” when setting and retrieving values. For example, the following syntax is valid:

```
oRequest( "ccAuthService_run" ) = "true"
```

The following example shows how to get the content of the Hashtable object as a string:

Example Using the Content Property of the Hashtable Object

```
content = oRequest.Content (vbCRLf)
```

The following example shows how to list the name-value pairs in the Hashtable object:

Example Using the Names Property of the Hashtable Object

```
names = oRequest.Names

For Each name in names

    Response.Write name & "=" & oRequest.Value(name)

Next
```

Example Using the Overwrite Property of the Hashtable Object

```
oRequest.Overwrite = true

oRequest.Value("name") = "valueA"

oRequest.Value("name") = "valueB"    ' overwrites valueA

oRequest.Overwrite = false

oRequest.Value("name") = "valueC"    ' does not overwrite anymore

                                     ' name=valueB
```

Hashtable Method

The Hashtable object has one method:

Delete

Table 8 Delete Method

Syntax	`Delete ( bstrName )`
Description	Deletes the entry for `bstrName` in the Hashtable, if it exists.
Returns	Nothing
Parameters	■ `bstrName`: The name for the name-value pair to delete.

Logger

Use the Logger object if you want to log more information than the client already does.

Important Logging can cause very large log files to accumulate. Therefore, CyberSource recommends that you use logging only when troubleshooting problems. To comply with all Payment Card Industry (PCI) and Payment Application (PA) Data Security Standards regarding the storage of credit card and card verification number data, the logs that are generated contain only masked credit card and card verification number data (CVV, CVC2, CVV2, CID, CVN).

Follow these guidelines:

- Use logging temporarily and for diagnostic purposes only.
- If possible, use logging only with test credit card numbers.
- Delete the log files as soon as you no longer need them.
- Never send email to CyberSource containing personal and account information, such as customers' names, addresses, card or check account numbers, and card verification numbers.

For more information about PCI and PABP requirements, see www.visa.com/cisp.

Logger Properties

The data type for all of the properties is string.

Table 9 Logger Properties

Property	Description
Filename	Filename to use for the log file. Field can include an absolute or relative path. For example, `c:\logs\file.log` or `..\file.log`. UNC paths are allowed. This property is read-only and can only be set by calling the PrepareFile method.
MaximumSize	Maximum size in megabytes for the log file. When the log file reaches this size, it is archived into <***logfilename***>.<***yyyymmdd***T***hhmmssxxx***> and a new log file is started. The ***xxx*** indicates milliseconds. The default value is `"10"`. This property is read-only and can only be set by calling the PrepareFile method.

Logger Methods

The Logger object has the following methods:

- PrepareFile
- Log
- LogTransactionStart

PrepareFile

Table 10 PrepareFile Method

Syntax	`PrepareFile( bstrFilename, bstrMaxSizeInMB )`
Description	Initializes the properties Filename and MaximumSize with `bstrFilename` and `bstrMaxSizeInMB`, respectively. The method then prepares the file by verifying that the file exceeds the maximum size. If it does, the method archives the file and creates a new empty file. You must call this method before calling the other Logger methods.
Returns	Nothing
Parameters	■ `bstrFilename`: The file name to use. ■ `bstrMaxSizeInMB`: Maximum size for the file.

Log

Table 11 Log Method

Syntax	`Log( bstrType, bstrText )`
Description	Writes an entry to the log file by using this format *`<timestamp> <thread id> <logType> > <logText>`* Example: 2003-11-07 22:47:12.484 1512 INFO > Missing field
Returns	Nothing
Parameters	■ `bstrType`: A short name describing the purpose of the log entry. For example, the client uses values such as REQUEST, REPLY, FAULT, INFO, TRANSTART, ERROR. To make the log file easy to view, CyberSource recommends that you use a maximum of 9 characters. ■ `bstrText`: The text to be logged in the file.

LogTransactionStart

Table 12 LogTransactionStart Method

Syntax	`LogTransactionStart( oMerchantConfig, oProxyConfig )`
Description	Writes an entry that marks the start of a transaction and that also logs the content of the supplied `oMerchantConfig` and `oProxyConfig` objects. Note that `oProxyConfig` can be Nothing, but `oMerchantConfig` must point to a MerchantConfig object. This method is optional but recommended. If you do not supply a Logger object in the request, the client creates its own Logger object and runs the LogTransactionStart method itself, creating a TRANSTART entry in the log file. If you supply a Logger object in the request, the client does not call this method on your Logger object, so you must call the method yourself before logging anything else in the log file. Alternately, you can choose some other way to indicate the start of the transaction in the log file.
Returns	Nothing
Parameters	▪ `oMerchantConfig`: The MerchantConfig object used for the transaction being logged. ▪ `oProxyConfig`: The ProxyConfig object used for the transaction being logged. Can be Nothing.

Fault

The Fault object contains information if a fault occurs when the client sends the request. The following table lists the Fault object's properties. The data type for all of the properties is string.

Table 13 Fault Properties

Property	Description
FaultDocument	The entire, unparsed fault document.
FaultCode	The fault code, which indicates where the fault originated.
FaultString	The fault string, which describes the fault.
RequestID	The requestID for the request.

Client

The Client object has one method.

RunTransaction

Table 14 RunTransaction Method

Syntax	`nStatus = RunTransaction ( oMerchantConfig, oProxyConfig, oLogger, varRequest, varReply, bstrErrorInfo)`
Description	Sends the request to the CyberSource server and receives the reply.
Returns	A number from 0 to 8 that indicates the status of the request. See "Possible Return Status Values," page 45 for descriptions of the values.
Parameters	▪ `oMerchantConfig`: A MerchantConfig object. Required. ▪ `oProxyConfig`: A ProxyConfig object. May be Nothing. ▪ `oLogger`: A "Logger" object. May be Nothing. Supply one only if you want to log more information than what the client logs. If you supply a Logger object, the client does not call its LogTransactionStart method. See "LogTransactionStart," page 44 for more information. ▪ `varRequest`: A variant that can be one of the items listed below in "Possible varRequest Values." ▪ `varReply`: A variant that can be one of the items listed below in "Possible varReply Values." ▪ `bstrErrorInfo`: If status is not 0, this parameter contains a BSTR that has more information about the status. If status is 0, this parameter is empty.

Possible Return Status Values

The runTransaction method returns a status indicating the result of the request. The following table describes the possible status values.

Table 15 Possible Status Values

Value	Description
0	**Result:** The client successfully received a reply. **Manual action to take:** None
1	**Result:** An error occurred before the request could be sent. This usually indicates a configuration problem with the client. **Manual action to take:** Fix the problem described in `bstrErrorInfo`.
2	**Result:** An error occurred while sending the request. **Manual action to take:** None.

Table 15 Possible Status Values (Continued)

Value	Description
3	**Result:** An error occurred while waiting for or retrieving the reply. **Manual action to take:** Check the Transaction Search screens on the Business Center to verify that the request was processed, and if so, whether it succeeded. Update your transaction database appropriately.
4	**Result:** The client received a reply or a fault, but an error occurred while processing it. **Value of varReply:** Contains the server's reply. **Manual action to take:** Examine the contents of `varReply`. If you cannot determine the status of the request, then check the Transaction Search screens on the Business Center to verify that the request was processed, and if so, whether it succeeded. Update your transaction database appropriately.
5	**Result:** The server returned a fault with FaultCode set to CriticalServerError. **Manual action to take:** Check the Transaction Search screens in the Business Center to verify that the request succeeded. When searching for the request, use the request ID provided in the RequestID property of the Fault object.
6	**Result:** The server returned a fault with FaultCode set to ServerError, indicating a problem with the CyberSource server. **Manual action to take:** None
7	**Result:** The server returned a fault with FaultCode set to a value other than ServerError or CriticalServerError. Indicates a possible problem with merchant status or the security key. Could also indicate that the message was tampered with after it was signed and before it reached the CyberSource server. **Manual action to take:** Examine the FaultString and fix the problem. You might need to generate a new security key, or you might need to contact Customer Support if there are problems with your merchant status.
8	**Result:** The server returned an HTTP status code other than 200 (OK) or 504 (gateway timeout). Note that if a 504 gateway timeout occurs, then the status=`3`. **Value of varReply:** Contains the HTTP response body, or if none was returned, the literal `"(response body unavailable)"`. **Manual action to take:** None.

Possible varRequest Values

The `varRequest` parameter can be one of the following:

- Hashtable object: Makes the client use the name-value pairs
- DOMDocument40 object (part of MSXML 4.0): Makes the client use XML
- BSTR containing XML data: Makes the client use XML

Possible varReply Values

The `varReply` parameter can be one of these:

- Hashtable object: If `varRequest` was a Hashtable object and the return status is `0`
- DOMDocument40 object: If `varRequest` was a DOMDocument40 object and the return status is `0`
- BSTR containing the XML reply: If `varRequest` was a BSTR and the return status is 0
- Fault: If the return status is `5`, `6`, or `7`. See "Fault," page 44 for information about the Fault object.
- BSTR containing the raw reply (the entire response body): If the return status is `4` or `8`
- Empty if the return status is `1`, `2`, or `3`

The following table summarizes the value of `varReply` that you receive for each status value. The `bstrErrorInfo` parameter in the reply always contains information about the status.

Table 16 Summary of varReply Values by Status

Status	0	1	2	3	4	5	6	7	8
DOMDocument40 or BSTR with XML or Hashtable (name-value pairs)	x								
BSTR with raw reply					x				x
Fault						x	x	x	

Using Name-Value Pairs

This section explains how to use the ASP client to request CyberSource services by using name-value pairs.

Requesting CyberSource Services

To request CyberSource services, write code that:

- Collects information for the CyberSource services that you will use
- Assembles the order information into requests
- Sends the requests to the CyberSource server
- Processes the reply information

Important The CyberSource servers do not support persistent HTTP connections.

The instructions in this section explain how to write VBScript that requests CyberSource services. For a list of API fields to use in your requests, see "Related Documents," page 21.

Creating and Sending Requests

Note The code in this section's example is incomplete. For a complete sample program, see the `AuthCaptureSample.wsf` file in the *`<installation directory>`*`\samples\ nvp` directory, or see the sample ASP pages.

To use any CyberSource service, you must create and send a request that includes the required information for that service.

The example that is developed in this section shows basic code for requesting CyberSource services. In this example, Jane Smith is buying an item for $29.95.

Creating the MerchantConfig Object

First create a MerchantConfig object and set your merchant ID and other basic transaction settings:

```
Dim oMerchantConfig
set oMerchantConfig =
    Server.CreateObject( "CyberSourceWS.MerchantConfig" )
oMerchantConfig.MerchantID = "infodev"
oMerchantConfig.KeysDirectory = "\keys"
oMerchantConfig.SendToProduction = "0"
oMerchantConfig.TargetAPIVersion = "1.18"
```

Creating an Empty Request Hashtable

You next create a Hashtable to hold the request fields:

```
Dim oRequest
set oRequest = Server.CreateObject( "CyberSourceWS.Hashtable" )
```

Adding the Merchant ID

You next add the CyberSource merchant ID to the request.

Note

If you specify a merchant ID in the Hashtable object, it overrides the merchant ID you specify in the MerchantConfig object.

```
oRequest.Value( "merchantID" ) = "infodev"
```

Adding Services to the Request Hashtable

You next indicate the service you want to use by adding the field to the request. For example, to request a credit card authorization:

```
oRequest.Value( "ccAuthService_run" ) = "true"
```

Requesting a Sale

You can request multiple services by adding additional fields to the request. For example, if you fulfill the order immediately, you can request credit card authorization and capture together (referred to as a “sale”):

```
oRequest.Value( "ccAuthService_run" ) = "true"
oRequest.Value( "ccCaptureService_run" ) = "true"
```

Adding Service-Specific Fields to the Request Hashtable

You next add the fields that are used by the services that you are requesting. If you request multiple services and they share common fields, you must add the field once only.

```
oRequest.Value( "billTo_firstName" ) = "Jane"
oRequest.Value( "billTo_lastName" ) = "Smith"
oRequest.Value( "card_accountNumber" ) = "4111111111111111"
oRequest.Value( "item_0_unitPrice" ) = "29.95"
```

The example above shows only a partial list of the fields you must send. Refer to "Related Documents," page 21 for information about the guides that list all of the fields for the services that you are requesting.

Sending the Request

You next create a Client object and send the request:

```
dim oClient
set oClient = WScript.CreateObject( "CyberSourceWS.Client" )
dim varReply, nStatus, strErrorInfo
nStatus = oClient.RunTransaction( _
          oMerchantConfig, Nothing, Nothing, oRequest, varReply, _
          strErrorInfo )
```

Interpreting Replies

Handling the Return Status

The `nStatus` handle is returned by the RunTransaction method. The `nStatus` indicates whether the CyberSource server received the request, the client received the reply, or there were any errors or faults during transmission. See "Possible Return Status Values," page 45 for descriptions of each status value. For a different example, see the `AuthCaptureSample.wsf` file in the `<`*`installation directory`*`>\samples\nvp` directory.

```
if nStatus = 0 then
  dim strContent
  ' Read the value of the "decision" in the varReply hashtable.
  ' If decision=ACCEPT, indicate to the customer that the request was successful.
  ' If decision=REJECT, indicate to the customer that the order was not approved.
  ' If decision=ERROR, indicate to the customer an error occurred and to try again
  ' later.
  strContent = GetReplyContent( varReply ) ' get reply contents
  ' See "Processing the Reason Codes," page 53 for how to process the reasonCode
  ' from the reply.
  ' Note that GetReplyContent() is included in this document to help you understand
  ' how to process reason codes, but it is not included as part of the sample scripts
  ' or sample ASP pages.

else HandleError nStatus, strErrorInfo, oRequest, varReply
end if
'---------------------
sub HandleError( nStatus, strErrorInfo, oRequest, varReply )
'---------------------
  ' HandleError shows how to handle the different errors that can occur.
  select case nStatus
    ' An error occurred before the request could be sent.
    case 1
      ' Non-critical error.
      ' Tell customer the order could not be completed and to try again later.
      ' Notify appropriate internal resources of the error.

    ' An error occurred while sending the request.
    case 2
      ' Non-critical error.
      ' Tell customer the order could not be completed and to try again later.

    ' An error occurred while waiting for or retrieving the reply.
    case 3
      ' Critial error.
      ' Tell customer the order could not be completed and to try again later.
      ' Notify appropriate internal resources of the error.

    ' An error occurred after receiving and during processing of the reply.
    case 4
      ' Critical error.
      ' Tell customer the order could not be completed and to try again later.
      ' Look at the BSTR in varReply for the raw reply.
      ' Notify appropriate internal resources of the error.
    ' CriticalServerError fault
    case 5
      ' Critial error.
      ' Tell customer the order could not be completed and to try again later.
      ' Read the contents of the Fault object in varReply.
      ' Notify appropriate internal resources of the fault.
    ' ServerError fault
    case 6
      ' Non-critical error.
      ' Tell customer the order could not be completed and to try again later.
      ' Read the contents of the Fault object in varReply.

    ' Other fault
    case 7
      ' Non-critical error.
      ' Tell customer the order could not be completed and to try again later.
      ' Read the contents of the Fault object in varReply.
      ' Notify appropriate internal resources of the fault.

    ' HTTP error
    Case 8
      ' Non-critical error.
      ' Tell customer the order could not be completed and to try again later.
      ' Look at the BSTR in varReply for the raw reply.

  end select
```

Processing the Reason Codes

After the CyberSource server processes your request, it sends a reply message that contains information about the services you requested. You receive different fields depending on the services you request and the outcome of each service.

To use the reply information, you must integrate it into your system and any other system that uses that data. For example, you can store the reply information in a database and send it to other back office applications.

You must write an error handler to process the reply information that you receive from CyberSource. Do not show the reply information directly to customers. Instead, present an appropriate response that tells customers the result.

Important Because CyberSource may add reply fields and reason codes at any time, you should parse the reply data according to the names of the fields instead of their order in the reply.

The most important reply fields to evaluate are the following:

- **decision**: A one-word description of the results of your request. The decision is one of the following:
  - `ACCEPT` if the request succeeded
  - `REJECT` if one or more of the services in the request was declined
  - `REVIEW` if you are a CyberSource Advanced merchant using CyberSource Decision Manager and it flags the order for review. See "Handling Decision Manager Reviews," page 55 for more information.
  - `ERROR` if there was a system error. See "Retrying When System Errors Occur," page 57 for more information.
- **reasonCode**: A numeric code that provides more specific information about the results of your request.

You also receive a reason code for each service in your request. You can use these reason codes to determine whether a specific service succeeded or failed. If a service fails, other services in your request may not run. For example, if you request a credit card authorization and capture, and the authorization fails, the capture does not run. The reason codes for each service are described in the *Credit Card Services User Guide* (for CyberSource Essentials merchants) or in the service's developer guide (for CyberSource Advanced merchants).

Important CyberSource reserves the right to add new reason codes at any time. If your error handler receives a reason code that it does not recognize, it should use the decision to interpret the reply.

```
' Note that GetReplyContent() is included in this document to help you
' understand how to process reason codes, but it is not included as part of
' the sample scripts or sample ASP pages.
'----------------
function GetReplyContent( varReply )
'----------------

  dim strReasonCode

  strReasonCode = varReply.Value( "reasonCode" )

  select case strReasonCode

    ' Success
    case "100"
      GetReplyContent _
      = "Request ID: " & varReply.Value( "requestID" ) & vbCrLf & _
      "Authorized Amount: " & _
      varReply.Value( "ccAuthReply_amount" ) & vbCrLf & _
      "Authorization Code: " & _
      varReply.Value( "ccAuthReply_authorizationCode" )

    ' Insufficient funds
    case "204"
      GetReplyContent = "Insufficient funds in account. Please use a different" & _
      "card or select another form of payment."

    ' add other reason codes here that you must handle specifically

    ' For all other reason codes, return an empty string, in which case, you should
    ' display a generic message appropriate to the decision value you received.
    case else
      GetReplyContent = ""

  end select

end function
```

Handling Decision Manager Reviews

If you use CyberSource Decision Manager, you may also receive the `REVIEW` value in the **decision** field. `REVIEW` means that Decision Manager has marked the order for review based on how you configured the Decision Manager rules.

If you will be using Decision Manager, you have to determine how to handle the new `REVIEW` value. Ideally, you will update your order management system to recognize the `REVIEW` response and handle it according to your business rules. If you cannot update your system to handle the `REVIEW` response, CyberSource recommends that you choose one of these options:

- If you authorize and capture the credit card payment at the same time, treat the `REVIEW` response like a `REJECT` response. Rejecting any orders that are marked for review may be appropriate if your product is a software download or access to a Web site. If supported by your processor, you may also want to reverse the authorization.

- If you approve the order after reviewing it, convert the order status to `ACCEPT` in your order management system. You can request the credit card capture without requesting a new authorization.

- If you approve the order after reviewing it but cannot convert the order status to `ACCEPT` in your system, request a new authorization for the order. When processing this new authorization, you must disable Decision Manager. Otherwise the order will be marked for review again. For details about the API field that disables Decision Manager, see the *Decision Manager Developer Guide Using the Simple Order API* (PDF | HTML) or the *Decision Manager Developer Guide Using the SCMP Order API* (PDF | HTML).

Alternately, you can specify a custom business rule in Decision Manager so that authorizations originating from a particular internal IP address at your company are automatically accepted.

If supported by your processor, you may want to reverse the original authorization.

Requesting Multiple Services

When you request multiple services in one request, CyberSource processes the services in a specific order. If a service fails, CyberSource does not process the subsequent services in the request.

For example, in the case of a sale (a credit card authorization and a capture requested together), if the authorization service fails, CyberSource will not process the capture service. The reply you receive only includes reply fields for the authorization.

The following example applies **to CyberSource Advanced merchants only.**

Many CyberSource services include “ignore” fields that tell CyberSource to ignore the result from the first service when deciding whether to run the subsequent services. In the case of the sale, even though the issuing bank gives you an authorization code, CyberSource might decline the authorization based on the AVS or card verification results. Depending on your business needs, you might choose to capture these types of declined authorizations anyway. You can set the **businessRules_ignoreAVSResult** field to “true” in your combined authorization and capture request:

```
oRequest.Value( "businessRules_ignoreAVSResult") = "true"
```

This tells CyberSource to continue processing the capture even if the AVS result causes CyberSource to decline the authorization. In this case you would then get reply fields for both the authorization and the capture in your reply.

Note

You are charged only for the services that CyberSource performs.

Retrying When System Errors Occur

You must design your transaction management system to include a way to correctly handle CyberSource system errors. Depending on which payment processor is handling the transaction, the error may indicate a valid CyberSource system error, or it may indicate a processor rejection because of some type of invalid data. In either case, CyberSource recommends that you do not design your system to retry sending a transaction many times in the case of a system error.

Instead, CyberSource recommends that you retry sending the request only two or three times with successively longer periods of time between each retry. For example, after the first system error response, wait 30 seconds and then retry sending the request. If you receive the same error a second time, wait one minute before you send the request again. Depending on the situation, you may decide you can retry sending the request after a longer time period. Determine what is most appropriate for your business situation.

If after several retry attempts you are still receiving a system error, it is possible that the error is actually being caused by a processor rejection and not a CyberSource system error. In that case, CyberSource recommends that you either:

- Search for the transaction in the Business Center, look at the description of the error on the Transaction Detail page, and call your processor to determine if and why they are rejecting the transaction.
- Contact CyberSource Customer Support to confirm whether your error is truly caused by a CyberSource system issue.

If TSYS Acquiring Solutions is your processor, you may want to follow the first suggestion as there are several common TSYS Acquiring Solutions processor responses that are returned to you as system errors and that only TSYS Acquiring Solutions can address.

Using XML

This section describes how to request CyberSource services using XML.

Requesting CyberSource Services

To request CyberSource services, write code that:

- Collects information for the services that you will use
- Assembles the order information into requests
- Sends the requests to the CyberSource server
- Processes the reply information

Important The CyberSource servers do not support persistent HTTP connections.

The instructions in this section explain how to write VBScript that requests CyberSource services. For a list of API fields to use in your requests, see "Related Documents," page 21.

Sample Code

CyberSource recommends that you examine the name-value pair sample code provided in `AuthCaptureSample.wsf` before implementing your code to process XML requests. The sample will give you a basic understanding of how to request CyberSource services. The sample code file is located in the *<installation directory>*`\samples\nvp` directory.

After examining that sample code, read this section to understand how to create code to process XML requests. Note that the code in this section's example is incomplete. For a complete sample program, see the `AuthSample.wsf` file in the *<installation directory>*`\samples\xml` directory.

Creating a Request Document

With the client, you can create an XML request document by using any application and send a request to CyberSource. For example, if you have a customer relationship management (CRM) system that uses XML to communicate with other systems, you can use the CRM system to generate request documents.

The XML document that you provide must be either a DOMDocument40 object (part of MSXML 4.0) or a string containing XML data.

The request document must be validated against the XML schema for CyberSource transactions. To view the `xsd` file for the version of the Simple Order API that you are using, go to https://ics2ws.ic3.com/commerce/1.x/transactionProcessor.

Important

Make sure that the elements in your document appear in the correct order. Otherwise, your document will not be valid, and your request will fail.

The following example shows a basic XML document for requesting CyberSource services. In this example, Jane Smith is buying an item for $29.95.

The XML document in this example is incomplete. For a complete example, see auth.xml in `<installation directory>\samples\xml`.

Creating an Empty Request

Add the XML declaration and the document's root element:

```
<?xml version="1.0" encoding="utf-8"?>
<requestMessage xmlns="urn:schemas-cybersource-com:transaction-data-
1.18">
</requestMessage>
```

When you construct a request, you must indicate the correct namespace for the elements, and the namespace must use the same API version that you specify in the MerchantConfig object. For example, if you set `TargetAPIVersion=1.18`, the namespace must be `urn:schemas-cybersource-com:transaction-data-1.18`.

Note

The XML document that you receive in the reply always uses a prefix of `c:` (for example, `xmlns:c="urn:schemas-cybersource-com:transaction-data-1.18"`). Make sure you use an XML parser that supports namespaces.

Adding the Merchant ID

You next add the CyberSource merchant ID to the request.

Note

If you specify a merchant ID in the XML document, it overrides the merchant ID you specify in the MerchantConfig object.

```
<?xml version="1.0" encoding="utf-8"?>
<requestMessage xmlns="urn:schemas-cybersource-com:transaction-data-
1.18">
  <merchantID>infodev</merchantID>
</requestMessage>
```

Adding Services to the Request

You next indicate the service that you want to use by creating an element for that service in the request, then setting the element's `run` attribute to `true`. For example, to request a credit card authorization:

```
<?xml version="1.0" encoding="utf-8"?>
<requestMessage xmlns="urn:schemas-cybersource-com:transaction-data-
1.18">
  <merchantID>infodev</merchantID>
  <ccAuthService run="true"/>
</requestMessage>
```

Requesting a Sale

You can request multiple services by adding additional elements. For example, if you fulfill the order immediately, you can request a credit card authorization and capture together (referred to as a "sale"):

```
<?xml version="1.0" encoding="utf-8"?>
<requestMessage xmlns="urn:schemas-cybersource-com:transaction-data-
1.18">
  <merchantID>infodev</merchantID>
  <ccAuthService run="true"/>
  <ccCaptureService run="true"/>
</requestMessage>
```

Adding Service-Specific Fields to the Request

You next add the fields that are used by the services you are requesting. Most fields are child elements of container elements; for example, a `<card>` element contains the customer's credit card information.

```
<?xml version="1.0" encoding="utf-8"?>
<requestMessage xmlns="urn:schemas-cybersource-com:transaction-data-
1.18">
  <merchantID>infodev</merchantID>
  <billTo>
    <firstName>Jane</firstName>
    <lastName>Smith</lastName>
  </billTo>
  <item id="0">
    <unitPrice>29.95</unitPrice>
  </item>
  <card>
    <accountNumber>4111111111111111</accountNumber>
  </card>
  <ccAuthService run="true"/>
</requestMessage>
```

The example above shows only a partial list of the fields you must send. Refer to "Related Documents," page 21 for information about the guides that list all of the fields for the services that you are requesting.

Sending Requests

Once you have created an XML document, you use VBscript in your ASP pages to send the request to CyberSource.

Creating the MerchantConfig Object

First create a MerchantConfig object and set your merchant ID and other basic transaction settings:

```
Dim oMerchantConfig
set oMerchantConfig =
    Server.CreateObject( "CyberSourceWS.MerchantConfig" )
oMerchantConfig.MerchantID = "infodev"
oMerchantConfig.KeysDirectory = "\keys"
oMerchantConfig.SendToProduction = "0"
oMerchantConfig.TargetAPIVersion = "1.18"
```

Note The namespace that you specify in the XML document must use the same API version that you specify in the MerchantConfig object. For example, if you set `TargetAPIVersion=1.18`, the namespace must be `urn:schemas-cybersource-com:transaction-data-1.18`.

Reading the XML Document

```
' read XML document
dim strXMLRequest
strXMLRequest = ReadTextFile( "MyXMLDocument.xml" )

' If you did not hardcode the namespace in the XML document, make sure
' to insert the effective namespace URI from the MerchantConfig object
' into the XML document.
```

Sending the Request

You next create a Client object and send the request:

```
dim oClient
set oClient = WScript.CreateObject( "CyberSourceWS.Client" )

' This example uses a DOMDocument40 object and not a string.
' See the AuthSample.wsf example in the samples\xml directory for an example
' of how to handle a string.
' Create the DOMDocument40 object
dim varRequest
set varRequest = WScript.CreateObject( "Msxml2.DOMDocument.4.0" )

' Load the XML string into the DOMDocument40 object
varRequest.loadXML( strXMLRequest )

' send request
dim varReply, nStatus, strErrorInfo
nStatus = oClient.RunTransaction( _
          oMerchantConfig, Nothing, Nothing, varRequest, _
          varReply, strErrorInfo )
```

Interpreting Replies

Handling the Return Status

The nStatus is the handle returned by the RunTransaction method. The nStatus indicates whether the CyberSource server received the request, the client received the reply, or there were any errors or faults during transmission. See "Possible Return Status Values," page 45 for descriptions of each status value. For a different example, see the AuthSample.wsf file in the client's *<installation directory>*\samples\xml directory.

```
' set the SelectionNamespace for later XPath queries
strNamespaceURI = oMerchantConfig.EffectiveNamespaceURI
varReply.setProperty "SelectionNamespaces", _
                     "xmlns:c='" & strNamespaceURI & "'"
if nStatus = 0 then
  dim strContent
  ' Get the contents of the <replyMessage> element
  dim oReplyMessage
  set oReplyMessage =
    varReply.SelectSingleNode( "c:replyMessage")
  ' Read the value of the "decision" in the oReplyMessage.
  ' If decision=ACCEPT, indicate to the customer that the request was successful.
  ' If decision=REJECT, indicate to the customer that the order was not approved.
  ' If decision=ERROR, indicate to the customer an error occurred and to try again
  ' later.
  ' Now get reason code results:
  strContent = GetReplyContent( oReplyMessage )
  ' See "Processing the Reason Codes," page 53 for how to process the reasonCode
  ' from the reply.
  ' Note that GetReplyContent() is included in this document to help you understand
  ' how to process reason codes, but it is not included as part of the sample scripts
  ' or sample ASP pages.
else HandleError nStatus, strErrorInfo, oRequest, varReply
end if
'---------------------
sub HandleError( nStatus, strErrorInfo, oRequest, varReply )
'---------------------
  ' HandleError shows how to handle the different errors that can occur.
  select case nStatus

    ' An error occurred before the request could be sent.
    case 1
      ' Non-critical error.
      ' Tell customer the order could not be completed and to try again later.
      ' Notify appropriate internal resources of the error.
```

```
     ' An error occurred while sending the request.
     case 2
       ' Non-critical error.
       ' Tell customer the order could not be completed and to try again later.
     ' An error occurred while waiting for or retrieving the reply.
     case 3
       ' Critical error.
       ' Tell customer the order could not be completed and to try again later.
       ' Notify appropriate internal resources of the error.

     ' An error occurred after receiving and during processing
     ' of the reply.
     case 4
       ' Critial error.
       ' Tell customer the order could not be completed and to try again later.
       ' Look at the BSTR in varReply for the raw reply.
       ' Notify appropriate internal resources of the error.
     ' CriticalServerError fault
     case 5
       ' Critial error.
       ' Tell customer the order could not be completed and to try again later.
       ' Read the contents of the Fault object in varReply.
       ' Notify appropriate internal resources of the fault.

     ' ServerError fault
     case 6
       ' Non-critical error.
       ' Tell customer the order could not be completed and to try again later.
       ' Read the contents of the Fault object in varReply.

     ' Other fault
     case 7
       ' Non-critical error.
       ' Tell customer the order could not be completed and to try again later.
       ' Read the contents of the Fault object in varReply.
       ' Notify appropriate internal resources of the fault.

     ' HTTP error
     Case 8
       ' Non-critical error.
       ' Tell customer the order could not be completed and to try again later.
       ' Look at the BSTR in varReply for the raw reply.

 end select
```

Processing the Reason Codes

After the CyberSource server processes your request, it sends a reply message that contains information about the services you requested. You receive different fields depending on the services you request and the outcome of each service.

To use the reply information, you must integrate it into your system and any other system that uses that data. For example, you can store the reply information in a database and send it to other back office applications.

You must write an error handler to process the reply information that you receive from CyberSource. Do not show the reply information directly to customers. Instead, present an appropriate response that tells customers the result.

Important

Because CyberSource may add reply fields and reason codes at any time, you should parse the reply data according to the names of the fields instead of their order in the reply.

The most important reply fields to evaluate are the following:

- **decision**: A one-word description of the results of your request. The decision is one of the following:
  - `ACCEPT` if the request succeeded
  - `REJECT` if one or more of the services in the request was declined
  - `REVIEW` if you use CyberSource Decision Manager and it flags the order for review. See "Handling Decision Manager Reviews," page 67 for more information.
  - `ERROR` if there was a system error. See "Retrying When System Errors Occur," page 57 for more information.
- **reasonCode**: A numeric code that provides more specific information about the results of your request.

You also receive a reason code for each service in your request. You can use these reason codes to determine whether a specific service succeeded or failed. If a service fails, other services in your request may not run. For example, if you request a credit card authorization and capture, and the authorization fails, the capture does not run. The reason codes for each service are described in the *Credit Card Services User Guide* (for CyberSource Essentials merchants) or in the service's developer guide (for CyberSource Advanced merchants).

Important

CyberSource reserves the right to add new reason codes at any time. If your error handler receives a reason code that it does not recognize, it should use the decision to interpret the reply.

```
' Note that GetReplyContent() is included in this document to help you
' understand how to process reason codes, but it is not included as
' part of the sample scripts or sample ASP pages.
'----------------
function GetReplyContent( oReplyMessage )
'----------------
  dim strReasonCode
  strReasonCode = GetValue( oReplyMessage, "c:reasonCode" )

  select case strReasonCode
    ' Success
    case "100"
      GetReplyContent = _
        "Request ID: " & GetValue( oReplyMessage, "c:requestID" ) & _
            vbCrLf & _
        "Authorized Amount: " & _
        GetValue( oReplyMessage, "c:ccAuthReply/c:amount" ) & _
            vbCrLf & _
        "Authorization Code: " & _
        GetValue( oReplyMessage, _
            "c:ccAuthReply/c:authorizationCode" )

    ' Insufficient funds
    case "204"
      GetReplyContent = "Insufficient funds in account. Please " & _
      "use a different card or select another form of payment." & _
    ' add other reason codes here that you must handle specifically

    ' For all other reason codes, return an empty string, in which case, you should
    ' display a generic message appropriate to the decision value you received.
    case else
      GetReplyContent = ""
  end select
end function
```

Handling Decision Manager Reviews

If you use CyberSource Decision Manager, you may also receive the `REVIEW` value in the **decision** field. `REVIEW` means that Decision Manager has marked the order for review based on how you configured the Decision Manager rules.

If you will be using Decision Manager, you have to determine how to handle the new `REVIEW` value. Ideally, you will update your order management system to recognize the `REVIEW` response and handle it according to your business rules. If you cannot update your system to handle the `REVIEW` response, CyberSource recommends that you choose one of these options:

- If you authorize and capture the credit card payment at the same time, treat the `REVIEW` response like a `REJECT` response. Rejecting any orders that are marked for review may be appropriate if your product is a software download or access to a Web site. If supported by your processor, you may also want to reverse the authorization.

- If you approve the order after reviewing it, convert the order status to `ACCEPT` in your order management system. You can request the credit card capture without requesting a new authorization.

- If you approve the order after reviewing it but cannot convert the order status to `ACCEPT` in your system, request a new authorization for the order. When processing this new authorization, you must disable Decision Manager. Otherwise the order will be marked for review again. For details about the API field that disables Decision Manager, see the *Decision Manager Developer Guide Using the Simple Order API* (PDF | HTML) or the *Decision Manager Developer Guide Using the SCMP Order API* (PDF | HTML).

Alternately, you can specify a custom business rule in Decision Manager so that authorizations originating from a particular internal IP address at your company are automatically accepted.

If supported by your processor, you may want to reverse the original authorization.

Requesting Multiple Services

When you request multiple services in one request, CyberSource processes the services in a specific order. If a service fails, CyberSource does not process the subsequent services in the request.

For example, in the case of a sale (a credit card authorization and a capture requested together), if the authorization service fails, CyberSource will not process the capture service. The reply you receive only includes reply fields for the authorization.

This following additional example applies **to CyberSource Advanced merchants only.**

Many CyberSource services include “ignore” fields that tell CyberSource to ignore the result from the first service when deciding whether to run the subsequent services. In the case of the sale, even though the issuing bank gives you an authorization code, CyberSource might decline the authorization based on the AVS or card verification results. Depending on your business needs, you might choose to capture these types of declined authorizations anyway. You can set the **businessRules_ignoreAVSResult** field to “true” in your combined authorization and capture request:

```
<businessRules>
    <ignoreAVSResult>true</ignoreAVSResult>
</businessRules>
```

This tells CyberSource to continue processing the capture even if the AVS result causes CyberSource to decline the authorization. In this case you would then get reply fields for both the authorization and the capture in your reply.

Note You are charged only for the services that CyberSource performs.

Retrying When System Errors Occur

You must design your transaction management system to include a way to correctly handle CyberSource system errors. Depending on which payment processor is handling the transaction, the error may indicate a valid CyberSource system error, or it may indicate a processor rejection because of some type of invalid data. In either case, CyberSource recommends that you do not design your system to retry sending a transaction many times in the case of a system error.

Instead, CyberSource recommends that you retry sending the request only two or three times with successively longer periods of time between each retry. For example, after the first system error response, wait 30 seconds and then retry sending the request. If you receive the same error a second time, wait one minute before you send the request again. Depending on the situation, you may decide you can retry sending the request after a longer time period. Determine what is most appropriate for your business situation.

If after several retry attempts you are still receiving a system error, it is possible that the error is actually being caused by a processor rejection and not a CyberSource system error. In that case, CyberSource recommends that you either:

- Search for the transaction in the Business Center, look at the description of the error on the Transaction Detail page, and call your processor to determine if and why they are rejecting the transaction.
- Contact CyberSource Customer Support to confirm whether your error is truly caused by a CyberSource system issue.

If TSYS Acquiring Solutions is your processor, you may want to follow the first suggestion as there are several common TSYS Acquiring Solutions processor responses that are returned to you as system errors and that only TSYS Acquiring Solutions can address.

CHAPTER

C/C++ Client

3

Important

- The C/C++ client for the Simple Order API is supported on 32-bit operating systems only.
- If you are building an application to sell to others, see Appendix A, "Using the Client Application Fields," on page 390. This appendix has a list of API fields you can use in your request that describe the application, its version, and its user. If you use these fields in your request, you can view their values in the Transaction Search Details window of the Business Center.

Choosing Your API and Client

API Variation

With this client package, you can use either of these variations of the Simple Order API:

- Name-value pairs, which are simpler to use than XML
- XML, which requires you to create and parse XML documents

The test that you run immediately after installing the client uses name-value pairs.

Client Versions

CyberSource updates the Simple Order API on a regular basis to introduce new API fields and functionality. To identify the latest version of the API, go to https://ics2wsa.ic3.com/commerce/1.x/transactionProcessor.

This represents the version of the server-side code for the CyberSource services.

The Simple Order API Client for C/C++ also has a version, but it is not the same as the API version. The client version represents the version of the client-side code that you use to access the CyberSource services.

When configuring the client, you indicate which version of the API you want to use. When setting this parameter, do not use the current version of the client; use the current version of the API.

Sample Code

The client contains two sets of sample code, one for using name-value pairs and one for using XML. See "Testing the Client," page 77 or see the README file for more information about using the sample code to test the client.

- Name-value pairs: See `authCaptureSample.c` in *<installation directory>*`/samples/nvp`.
- XML: We suggest that you examine the name-value pair sample code listed above before implementing your code to process XML requests.

For the XML sample code, see `authSample.c` in *<installation directory>*`/samples/xml`. Also see the `auth.xml` XML document that the script uses.

Basic C/C++ Page Example

The following example shows the code required to send a Simple Order API request for credit card authorization and process the reply. The example uses name-value pairs. For a more complete example, see the sample code and sample store included in the package (see "Sample Code," page 71). "Using Name-Value Pairs," page 90 shows you how to create the code.

```
#include "cybersource.h"

// Load the configuration settings

const char CYBS_CONFIG_INI_FILE[] = "../cybs.ini";
pConfig = cybs_load_config( CYBS_CONFIG_INI_FILE );

// Set up the request by creating an empty CybsMap and add fields to it

pRequest = cybs_create_map();

// We want to do credit card authorization in this example

cybs_add( pRequest, "ccAuthService_run", "true" );
```

```
// Add required fields

cybs_add( pRequest, "merchantID", "infodev" );
cybs_add( pRequest, "merchantReferenceCode", "MRC-14344" );
cybs_add( pRequest, "billTo_firstName", "Jane" );
cybs_add( pRequest, "billTo_lastName", "Smith" );
cybs_add( pRequest, "billTo_street1", "Charleston" );
cybs_add( pRequest, "billTo_city", "Mountain View" );
cybs_add( pRequest, "billTo_state", "CA" );
cybs_add( pRequest, "billTo_postalCode", "94043" );
cybs_add( pRequest, "billTo_country", "US" );
cybs_add( pRequest, "billTo_email", "jsmith@example.com" );
cybs_add( pRequest, "card_accountNumber", "4111111111111111" );
cybs_add( pRequest, "card_expirationMonth", "12" );
cybs_add( pRequest, "card_expirationYear", "2010" );
cybs_add( pRequest, "purchaseTotals_currency", "USD" );

// This example has two items

cybs_add( pRequest, "item_0_unitPrice", "12.34" );
cybs_add( pRequest, "item_1_unitPrice", "56.78" );

// Add optional fields here according to your business needs
// Send request

Create the reply structure and send the request
pReply = cybs_create_map();
status = cybs_run_transaction(pConfig, pRequest, pReply);

// Handle the reply. See "Handling the Return Status," page 93.
```

Installing and Testing the Client

Minimum System Requirements

For Linux

- Linux kernel 2.2, LibC6 on an Intel processor
- GNU GCC compiler (with C++ enabled)

For Windows

- Windows XP, 2000, or newer
- Microsoft Visual Studio 6.0

The SDK supports UTF-8 encoding.

Important

Failure to configure your client API host to a unique, public IP address will cause inconsistent transaction results.

The client API request ID algorithm uses a combination of IP address and system time, along with other values. In some architectures this combination might not yield unique identifiers.

Transaction Security Keys

The first thing you must do is create your security key. The client uses the security key to add a digital signature to every request that you send. This signature helps ensure that no one else can use your CyberSource account to process orders. You specify the location of your key when you configure the client.

Important You must generate two transaction security keys—one for the CyberSource production environment and one for the test environment. For information about generating and using security keys, see *Creating and Using Security Keys* (PDF | HTML).

The Simple Order API client for C/C++ package includes the `ca-bundle.crt`, a bundle of certificate files. The client expects to find the `ca-bundle.crt` file in the same directory as your security keys. If you decide to move it elsewhere, use the sslCertFile configuration parameter to specify the file's location (see the description of "sslCertFile," page 76).

Warning You must protect your security key to ensure that your CyberSource account is not compromised.

Installing the Client

To install the client:

Step 1 Go to the client downloads page on the Support Center.

Step 2 Download the latest client package, and save it in any directory.

Step 3 Unpack the file.

This creates an installation directory called `simapi-c-n.n.n`, where `n.n.n` is the client version. The client is now installed on your system.

Step 4 Configure the client. See "Configuring Client Settings" below.

Step 5 Test the client. See "Testing the Client," page 77.

You have installed and tested the client. You are ready to create your own code for requesting CyberSource services. Finish reading this section, and then move on to either "Using Name-Value Pairs," page 90 if you plan to use name-value pairs, or "Using XML," page 100 if you plan to use XML.

Configuring Client Settings

To run the sample code included in the client package, you must set the configuration parameters in the `cybs.ini` file, which is located in the installation directory. You can also use this file when running transactions in a production environment (see the function descriptions in "C/C++ API for the Client," page 80). Table 17 describes the parameters that you can set. Note that the default `cybs.ini` file that comes with the client package does not include all of the parameters listed in Table 17. It includes only the ones required to run the sample code.

If you have multiple merchant IDs, or if you are a reseller handling multiple merchants, you can use different configuration settings depending on the merchant ID. See "Configuring for Multiple Merchant IDs," page 115 for more information.

Table 17 Configuration Settings

Setting	Description
merchantID	Merchant ID. This client uses this value if you do not specify a merchant ID in the request itself.
keysDirectory	Location of the merchant's security keys for the production and the test environments. The client includes a `keys` directory that you can use. **Note** CyberSource recommends that you store your key locally for faster request processing.
sendToProduction	Flag that indicates whether the transactions for this merchant should be sent to the production server. Use one of these values: ▪ `false`: Do not send to the production server; send to the test server (default setting). ▪ `true`: Send to the production server.
targetAPIVersion	Version of the Simple Order API to use, for example: `1.18`. Do not set this property to the current version of the client; set it to an available API version. See "Client Versions," page 70 for more information. **Note** Go to https://ics2wsa.ic3.com/commerce/1.x/transactionProcessor to see a current list of the available versions. See the *Simple Order API Release Notes*, for information about what has changed in each version.
keyFilename	Name of the security key filename for the merchant in the format `<`***`key_fileName`***`>.p12`.
serverURL	Alternate server URL to use. See "Using Alternate Server Configuration Settings," page 114 for more information. Give the complete URL because it will be used exactly as you specify here.
namespaceURI	Alternate namespace URI to use. See "Using Alternate Server Configuration Settings," page 114 for more information. Give the complete namespace URI, as it will be used exactly as you specify here.

Table 17 Configuration Settings (Continued)

Setting	Description
enableLog	Flag directing the client to log transactions and errors. Possible values: ■ `false`: Do not enable logging (default setting). ■ `true`: Enable logging. **Important** Logging can cause very large log files to accumulate. Therefore, CyberSource recommends that you use logging only when troubleshooting problems. To comply with all Payment Card Industry (PCI) and Payment Application (PA) Data Security Standards regarding the storage of credit card and card verification number data, the logs that are generated contain only masked credit card and card verification number data (CVV, CVC2, CVV2, CID, CVN). Follow these guidelines: ■ Use debugging temporarily for diagnostic purposes only. ■ If possible, use debugging only with test credit card numbers. ■ Never store clear text card verification numbers. ■ Delete the log files as soon as you no longer need them. ■ Never send email to CyberSource containing personal and account information, such as customers' names, addresses, card or check account numbers, and card verification numbers. For more information about PCI and PABP requirements, see www.visa.com/cisp.
logDirectory	Directory to which to write the log file. Note that the client will not create this directory for you; you must specify an existing directory.The client includes a `logs` directory that you can use.
logFilename	Log file name. The client uses `cybs.log` by default.
logMaximumSize	Maximum size in megabytes for the log file. The default value is `"10"`. When the log file reaches the specified size, it is archived into `cybs.log.`***<yyyymmdd***`T`***hhmmssxxx>*** and a new log file is started. The ***xxx*** indicates milliseconds.
sslCertFile	The location of the bundled file of CA Root Certificates (`ca-bundle.crt`) which is included in the client download package. The client automatically looks for the file in the directory where your security keys are stored (specified by keysDirectory). If you move the file so it does not reside in keysDirectory, use this configuration setting to specify the full path to the file, including the file name.
timeout	Length of timeout in seconds. The default is 110.
proxyServer	Proxy server to use. Allowable formats include: ■ *<http://>server<:port>* ■ *<http://>IP address<:port>* The `http://` and `port` are optional. **Note** The default port is 1080. If your proxy server is listening on another port, you must specify a port number.

Table 17 Configuration Settings (Continued)

Setting	Description
proxyUsername	Username used to authenticate against the proxy server, if required. If the proxy server requires the domain name during authentication, add the domain name and a backslash: ***`<domain>`***`\`***`<username>`***
proxyPassword	Password used to authenticate against the proxy server, if required.

Testing the Client

After you install and configure the client, test it immediately to ensure that the installation is successful.

To test the client:

Step 1 At a command prompt, go to the `<installation directory>/samples/nvp` directory.

Step 2 Run the sample program by typing

`authCaptureSample`

The results of the test are displayed in the window.

- If the test is successful, a decision of ACCEPT appears for both the credit card authorization and the follow-on capture.
- If the test is not successful, a different decision value or an error message appears.

To troubleshoot if the test fails:

Step 1 Check to see that your `cybs.ini` settings are correct.

Step 2 Run the test again.

Step 3 If the test still fails, look at the error message and find the return status value (a numeric value from 0 to 8).

Step 4 See the descriptions of the status values in "Possible Return Status Values," page 86 and follow any instructions given there for the error you received.

Step 5 Run the test again.

Step 6 If the test still fails, contact Customer Support.

To run the XML sample:

Step 1 At a command prompt, go to the `<installation directory>/samples/xml` directory.

Step 2 Run the sample program by typing

```
authSample
```

The results of the test are displayed in the window.

- If the test is successful, a decision of ACCEPT appears for both the credit card authorization and the follow-on capture.
- If the test is not successful, a different decision value or an error message appears.

Going Live

When you have completed all of your system testing and are ready to accept real transactions from your customers, your deployment is ready to *go live*.

CyberSource Essentials Merchants

If you use CyberSource Essentials services, you can use the Business Center site to go live. For a description of the process of going live, see the "Steps for Getting Started" section in *Getting Started with CyberSource Essentials*.

Important You must also configure your client so that it sends transactions to the production server and not the test server. See the description of the `sendToProduction` setting in Table 17, page 75.

After your deployment goes live, use real card numbers and other data to test every card type you support. Because these are real transactions in which you are buying from yourself, use small monetary amounts to do the tests. Process an authorization, then capture the authorization, and later refund the money. Use your bank statements to verify that money is deposited into and withdrawn from your merchant bank account as expected. If you have more than one CyberSource merchant ID, test each one separately.

CyberSource Advanced Merchants

If you use CyberSource Advanced services, see the "Steps for Getting Started" chapter in *Getting Started with CyberSource Advanced* for information about going live.

When your deployment goes live, your CyberSource account is updated so that you can send transactions to the CyberSource production server. If you have not already done so, you must provide your banking information to CyberSource so that your processor can deposit funds to your merchant bank account.

After CyberSource confirms that your deployment is live, make sure that you update your system so that it can send requests to the production server (`ics2wsa.ic3.com`) using your security keys for the production environment. The test server (`ics2wstesta.ic3.com`) cannot be used for real transactions. For more information about sending transactions to the production server, see the description of the configuration setting "sendToProduction," page 75.

After your deployment goes live, use real card numbers and other data to test every card type, currency, and CyberSource application that your integration supports. Because these are real transactions in which you are buying from yourself, use small monetary amounts to do the tests. Use your bank statements to verify that money is deposited into and withdrawn from your merchant bank account as expected. If you have more than one CyberSource merchant ID, test each one separately.

Updating the Client to Use a Later API Version

CyberSource periodically updates the Simple Order API (previously called the Web Services API). You can update your existing client to work with the new API version. Go to https://ics2wsa.ic3.com/commerce/1.x/transactionProcessor for a list of the available API versions.

To update the client to use a later API version, update the value for the `targetAPIVersion` configuration parameter. For example, to use the 1.18 version of the API, set the property to `1.18`.

C/C++ API for the Client

CybsMap Structure

CybsMap is the structure that contains your configuration settings, your request, and the reply. You use the functions described in the next section to manipulate the structure, which includes adding the configuration settings, adding either name-value pairs or an XML document for the request, sending the request, and retrieving the corresponding reply.

Available Functions

The client API includes the functions described in these sections:

- "cybs_load_config()," page 81
- "cybs_create_map()," page 81
- "cybs_destroy_map()," page 81
- "cybs_set_add_behavior()," page 82
- "cybs_add()," page 82
- "cybs_remove()," page 82
- "cybs_get()," page 83
- "cybs_get_first()," page 83
- "cybs_get_next()," page 84
- "cybs_get_count()," page 84
- "cybs_create_map_string()," page 84
- "cybs_destroy_map_string()," page 85
- "cybs_run_transaction()," page 85

cybs_load_config()

Table 18 cybs_load_config()

Syntax	`CybsMap *cybs_load_config( const char *szFilename )`
Description	Creates an empty CybsMap structure and loads the configuration settings into the structure from a file. If you include a configuration property in the file more than once, the behavior is undefined. The `add behavior` setting (see "cybs_set_add_behavior()," page 82) of the returned map is set to 2 (overwrite). This allows you to use the cybs_add() function ("cybs_add()," page 82) to immediately override any settings that were read from the configuration file. You must later free the returned pointer by using cybs_destroy_map() (see "cybs_destroy_map()," page 81).
Returns	Returns a pointer to the CybsMap structure containing the configuration settings.
Parameters	`szFilename`: Name of the configuration file with the full or relative path.

cybs_create_map()

Table 19 cybs_create_map()

Syntax	`CybsMap *cybs_create_map()`
Description	Creates an empty CybsMap structure with the add behavior set to CYBS_NO_CHECK. You must later free the returned pointer by using cybs_destroy_map() (see "cybs_destroy_map()," page 81).
Returns	Returns a pointer to the new empty CybsMap structure.
Parameters	None.

cybs_destroy_map()

Table 20 cybs_destroy_map()

Syntax	`void cybs_destroy_map( CybsMap *pMap )`
Description	Destroys a CybsMap structure created with either cybs_create_map() or cybs_load_config().
Returns	Returns nothing.
Parameters	`pMap`: The CybsMap structure to be destroyed.

cybs_set_add_behavior()

Table 21 cybs_set_add_behavior()

Syntax	`CybsAddBehavior cybs_set_add_behavior(`
	`CybsMap *pRequest, CybsAddBehavior add_behavior )`

Description	Sets the type of add behavior that will be used when you add name-value pairs to the specified message structure: ▪ 0: When you add a new name-value pair, the client does not check to see if the name-value pair already exists in the structure. If the name already exists, the client still adds the name-value pair to the structure. This is the default value for cybs_create_map(). ▪ 1: If you try to add a name that already exists in the structure, the client keeps the existing name and value. The client does not allow you to add the same name or change the value of an existing name. ▪ 2: If you try to add a name that already exists in the structure, the client overwrites the existing name's value with the new value. This is the default value for cybs_load_config().
Returns	Returns the previous add behavior setting.
Parameters	`pRequest`: The CybsMap structure in which to apply the add behavior setting. `add_behavior`: The add behavior type to assign to the structure.

cybs_add()

Table 22 cybs_add()

Syntax	`int cybs_add( CybsMap *pRequest, const char *szName, const char *szValue )`
Description	Adds a name-value pair to the specified message structure.The function will do nothing if `pRequest`, `szName`, or `szValue` is null. With this function you can add name-value pairs for API fields or for configuration settings.
Returns	Returns `0` on success or `-1` on failure.
Parameters	`pRequest`: The CybsMap structure to add the name-value pairs to. `szName`: The name to add. `szValue`: The value to add.

cybs_remove()

Table 23 cybs_remove()

Syntax	`void cybs_remove( CybsMap *pRequest, const char *szName )`
Description	Uses the specified name to remove the name-value pair from the structure.
Returns	Returns nothing. Simply returns if the name does not exist.
Parameters	`pRequest`: The CybsMap structure to be used. `szName`: The name of the value to remove.

cybs_get()

Table 24 cybs_get()

Syntax	`const char *cybs_get( CybsMap *pMap, const char *szName )`
Description	Gets the value corresponding to the specified name. Note that this pointer is owned by the client and you should not free it.
Returns	Returns a pointer to the value or null if the name does not exist.
Parameters	`pMap`: The CybsMap structure to be used. `szName`: The name to use.

cybs_get_first()

Table 25 cybs_get_first()

Syntax	`void cybs_get_first( CybsMap *pMap, const char **pszName, const char **pszValue )`
Description	Returns a pointer to the first name and to its value in the map. Note that the entries in the map are not sorted in any way. If the map contains no entries, `*pszName` and `*pszValue` are null. It is sufficient just to check `*pszName`. Note that the pointers `*pszName` and `*pszValue` are owned by the client; you should not free them. Use cybs_get_next() to get the subsequent entries (see "cybs_get_next()," page 84).
Returns	Returns nothing.
Parameters	`pMap`: The CybsMap structure to use. `*pszName`: Pointer to the first name in the map. `*pszValue`: Pointer to the value of the first name in the map.

cybs_get_next()

Table 26 cybs_get_next()

Syntax	`void cybs_get_next( CybsMap *pMap, const char **pszName, const char **pszValue )`
Description	Returns a pointer to the next name and to its value in the map. Note that the entries in the map are not sorted in any way. You may use this function only after using cybs_get_first() with the same CybsMap structure. If the map contains no more entries, then `*pszName` and `*pszValue` would be null. It is sufficient just to check `*pszName`. Note that the pointers `*pszName` and `*pszValue` are owned by the client; you should not free them. The function's behavior is undefined if you update the map (for example, if you add a new entry) between calls to cybs_get_first() and cybs_get_next().
Returns	Returns nothing.
Parameters	`pMap`: The CybsMap structure to use. `*pszName`: Pointer to the first name in the map. `*pszValue`: Pointer to the value of the first name in the map.

cybs_get_count()

Table 27 cybs_get_count()

Syntax	`int cybs_get_count( CybsMap *pMap )`
Description	Returns the number of name-value pairs in the specified message structure.
Returns	Returns the number of name-value pairs.
Parameters	`pMap`: The CybsMap structure to use.

cybs_create_map_string()

Table 28 cybs_create_map_string()

Syntax	`char *cybs_create_map_string( CybsMap *pMap )`
Description	Creates a string containing all of the name-value pairs in the structure separated by the newline character sequence that is appropriate to the operating system. If the structure is empty, the function returns an empty string. On failure, the function returns null. You must later free the returned pointer using cybs_destroy_map_string() (see below).
Returns	Returns a pointer to the string containing the name-value pairs.
Parameters	`pMap`: The CybsMap structure to use.

cybs_destroy_map_string()

Table 29 cybs_destroy_map_string()

Syntax	void cybs_destroy_map_string(char *szMapString)
Description	Destroys a string created with cybs_create_map_string().
Returns	Returns nothing.
Parameters	szMapString: The map string to destroy.

cybs_run_transaction()

Table 30 cybs_run_transaction()

Syntax	CybsStatus cybs_run_transaction(CybsMap *pConfig, CybsMap *pRequest, CybsMap **ppReply)	
Description	Sends the request to the CyberSource server and receives the reply.	
Returns	A value that indicates the status of the request (see Table 31, page 86 for a list of values).	
Parameters	pconfig: Pointer to the configuration map structure to use.	
	pRequest: Pointer to a map structure containing one of these:	■ The individual name-value pairs in the request (for name-value pair users) ■ A single key called _xml_document whose value is the XML document representing the request (for XML users)
	ppReply: Pointer to a pointer to a map structure containing one of these:	■ The individual name-value pairs in the reply (for name-value pair users) ■ A single key called _xml_document whose value is the XML document representing the reply (for XML users) ■ If an error occurs, a combination of these keys and their values: _error_info _raw_reply _fault_document _fault_code _fault_string _fault_request_id See below for descriptions of these keys. **Note** You must later free the *ppReply pointer with cybs_destroy_map() (see "cybs_destroy_map()," page 81).

Reply Key Descriptions

- `_error_info`: Information about the error that occurred
- `_raw_reply`: The server's raw reply
- `_fault_document`: The entire, unparsed fault document
- `_fault_code`: The fault code, which indicates where the fault originated
- `_fault_string`: The fault string, which describes the fault.
- `_fault_request_id`: The request ID for the request.

Possible Return Status Values

The cybs_run_transaction() function returns a status indicating the result of the request. Table 31 describes the possible status values, including whether the error is critical. If an error occurs after the request has been sent to the server, but the client cannot determine whether the transaction was successful, then the error is considered critical. If a critical error happens, the transaction may be complete in the CyberSource system but not complete in your order system. The descriptions below indicate how to handle critical errors.

Note

The sample code (when run from a command prompt) displays a numeric value for the return status, which is listed in the first column.

Table 31 Possible Status Values

Numeric Value (for Sample Code)	Value	Description
0	CYBS_S_OK	**Critical:** No **Result:** The client successfully received a reply. **Keys in **ppReply:** For name-value pair users, **`ppReply` has the reply name-value pairs for the services that you requested. For XML users, **`ppReply` contains the `_xml_document` key, with the response in XML format. **Manual action to take:** None
1	CYBS_S_ PRE_SEND_ ERROR	**Critical:** No **Result:** An error occurred before the request could be sent. This usually indicates a configuration problem with the client. **Keys in **ppReply:** `_error_info` **Manual action to take:** Fix the problem described in the error information.

Table 31 Possible Status Values (Continued)

Numeric Value (for Sample Code)	Value	Description
2	CYBS_S_ SEND_ ERROR	**Critical:** No **Result:** An error occurred while sending the request. **Keys in **ppReply:** `_error_info` **Manual action to take:** None **Note** A typical send error that you might receive when testing occurs if the `ca-bundle.crt` file is not located in the same directory as your security key. See the description of the sslCertFile configuration parameter in Table 17, page 75 for information about how to fix the problem.

3	CYBS_S_ RECEIVE_ ERROR	**Critical:** Yes
		Result: An error occurred while waiting for or retrieving the reply.
		Keys in **ppReply:
		`_error_info`
		`_raw_reply`
		Manual action to take: Check the Transaction Search screens on the Business Center to verify that the request was processed, and if so, whether it succeeded. Update your transaction database appropriately.

4	CYBS_S_ POST_ RECEIVE_ ERROR	**Critical:** Yes
		Result: The client received a reply or a fault, but an error occurred while processing it.
		Keys in **ppReply:
		`_error_info`
		`_raw_reply`
		Manual action to take: Examine the value of `_raw_reply`. If you cannot determine the status of the request, then check the Transaction Search screens on the Business Center to verify that the request was processed, and if so, whether it succeeded. Update your transaction database appropriately.

Table 31 Possible Status Values (Continued)

Numeric Value (for Sample Code)	Value	Description

5	CYBS_S_CRITICAL_SERVER_FAULT	**Critical:** Yes
		Result: The server returned a fault with `_fault_code` set to CriticalServerError.
		Keys in **ppReply:
		`_error_info`
		`_fault_document`
		`_fault_code`
		`_fault_string`
		`_fault_request_id`
		Manual action to take: Check the Transaction Search screens on the Business Center to verify that the request succeeded. When searching for the request, use the request ID provided by `_fault_request_id`.

6	CYBS_S_SERVER_FAULT	**Critical:** No
		Result: The server returned a fault with `_fault_code` set to ServerError, indicating a problem with the CyberSource server.
		Keys in **ppReply:
		`_error_info`
		`_fault_document`
		`_fault_code`
		`_fault_string`
		Manual action to take: None

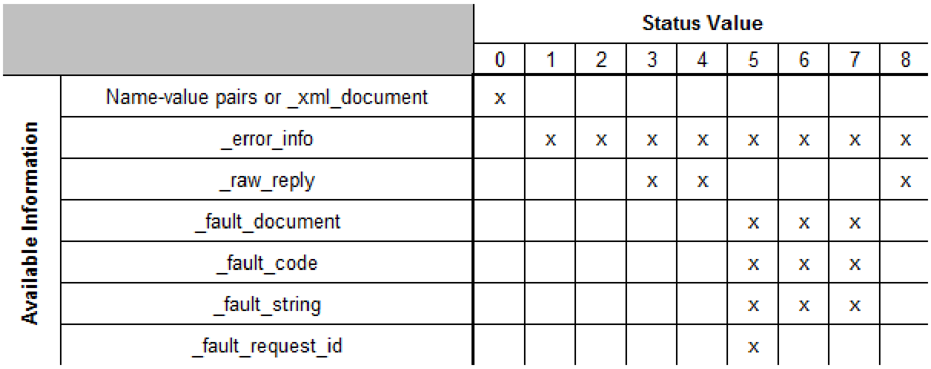
Table 31 Possible Status Values (Continued)

Numeric Value (for Sample Code)	Value	Description

7	CYBS_S_
	OTHER_
	FAULT
		Critical: No
		Result: The server returned a fault with `_fault_code` set to a value other than ServerError or CriticalServerError. Indicates a possible problem with merchant status or the security key. Could also indicate that the message was tampered with after it was signed and before it reached the CyberSource server.
		Keys in **ppReply:
		`_error_info`
		`_fault_document`
		`_fault_code`
		`_fault_string`
		Manual action to take: Examine the value of the `_fault_string` and fix the problem. You might need to generate a new security key, or you might need to contact Customer Support if there are problems with your merchant status. For more information, see *Creating and Using Security Keys* (PDF | HTML).
		Note A typical error that you might receive occurs if your merchant ID is configured for "test" mode but you send transactions to the production server. See the description of the sendToProduction configuration parameter in Table 17, page 75 for information about fixing the problem.

8	CYBS_S_
	HTTP_
	ERROR
		Critical: No
		Result: The server returned an HTTP status code other than 200 (OK) or 504 (gateway timeout). Note that if a 504 gateway timeout occurs, then the status=`3`.
		Keys in **ppReply:
		`_error_info`
		`_raw_reply` (contains the HTTP response body, or if none was returned, the literal `"(no response available)").`
		Manual action to take: None.

The figure below summarizes the reply information you receive for each status value.



		Status Value								
		0	1	2	3	4	5	6	7	8
Available Information	Name-value pairs or _xml_document	x								
	_error_info		x	x	x	x	x	x	x	x
	_raw_reply				x	x				x
	_fault_document						x	x	x	
	_fault_code						x	x	x	
	_fault_string						x	x	x	
	_fault_request_id						x			

Using Name-Value Pairs

This section explains how to use the client to request CyberSource services by using name-value pairs.

Requesting CyberSource Services

To request CyberSource services, write code that:

- Collects information for the CyberSource services that you will use
- Assembles the order information into requests
- Sends the requests to the CyberSource server
- Processes the reply information

Important The CyberSource servers do not support persistent HTTP connections.

The instructions in this section explain how to use C/C++ to request CyberSource services. For a list of API fields to use in your requests, see "Related Documents," page 21.

Sample Code

The code in this section's example is incomplete. For a complete sample program, see the `authCaptureSample.c` file in the `<`*`installation directory`*`>/samples/nvp` directory.

Creating and Sending Requests

To use any CyberSource service, you must create and send a request that includes the required information for that service.

The following example shows basic code for requesting CyberSource services. In this example, Jane Smith is buying an item for $29.95.

Adding the Use Statement

First add the include statement for the `cybersource.h` file:

```
#include "cybersource.h"
```

Loading the Configuration Settings

Next use cybs_load_config() to create a new CybsMap structure and load the configuration settings from a file:

```
const char CYBS_CONFIG_INI_FILE[] = "../cybs.ini";
pConfig = cybs_load_config( CYBS_CONFIG_INI_FILE );
```

You could instead create an empty CybsMap structure and add each configuration setting separately. You could also use a combination of the two methods: You could read the settings from a file and then add new settings using the cybs_add() function to override the settings read from the file.

Creating the Empty Request and Reply

Next use cybs_create_map() to create the request and reply:

```
pRequest = cybs_create_map();
pReply = cybs_create_map();
```

Adding the Merchant ID

You next add the CyberSource merchant ID to the request. You can let the CyberSource C/C++ client automatically retrieve the merchant ID from the pConfig structure, or you can set it directly in the request (see below). The pRequest value overrides the pConfig value.

```
cybs_add( pRequest, "merchantID",   "infodev" );
```

Adding Services to the Request Structure

You next indicate the service you want to use by adding the field to the request. For example, to request a credit card authorization:

```
cybs_add( pRequest, "ccAuthService_run",   "true" );
```

Requesting a Sale

You can request multiple services by adding additional fields to the request. For example, if you fulfill the order immediately, you can request credit card authorization and capture together (referred to as a “sale”):

```
cybs_add( pRequest, "ccAuthService_run",   "true" );
cybs_add( pRequest, "ccCaptureService_run",   "true" );
```

Adding Service-Specific Fields to the Request

You next add the fields that are used by the services that you are requesting. If you request multiple services and they share common fields, you must add the field once only.

```
cybs_add( pRequest, "merchantReferenceCode",   "3009AF229L7W" );
cybs_add( pRequest, "billTo_firstName",   "Jane" );
cybs_add( pRequest, "billTo_lastName",   "Smith" );
cybs_add( pRequest, "card_accountNumber",   "4111111111111111" );
cybs_add( pRequest, "item_0_unitPrice",   "29.95" );
```

The example above shows only a partial list of the fields you must send. Refer to "Requesting CyberSource Services," page 90 for information about the guides that list all of the fields for the services that you are requesting.

Sending the Request

You next send the request:

```
status = cybs_run_transaction( pConfig, pRequest, pReply );
```

Interpreting Replies

Handling the Return Status

The `status` value is the handle returned by the cybs_run_transaction() function. The `status` indicates whether the CyberSource server received the request, the client received the reply, or there were any errors or faults during transmission. See "Possible Return Status Values," page 86 for descriptions of each status value. For a different example, see the `authCaptureSample.c` file in the `<installation directory>/samples/nvp` directory:

```
if( status == CYBS_S_OK ) {

  // Read the value of the "decision" in pReply.

  decision = cybs_get( pReply, "decision" );

  // If decision=ACCEPT, indicate to the customer that the request was successful.
  // If decision=REJECT, indicate to the customer that the order was not approved.
  // If decision=ERROR, indicate to the customer that an error occurred and to try
  // again later.
  // Now get reason code results:

  reason = cybs_get( pReply, "reasonCode" );

  // See "Processing the Reason Codes," page 96 for how to process the
  // reasonCode from the reply.

} else {
  handleError( status, pRequest, pReply );
}

//---------------------

void handleError( CybsStatus stat, CybsMap* preq, CybsMap* prpl )

//---------------------

{

  // handleError shows how to handle the different errors that can occur.

  const char* pstr;
  pstr = cybs_get( prpl, CYBS_SK_ERROR_INFO );
  switch( stat ) {
```

```
// An error occurred before the request could be sent.

case CYBS_S_PRE_SEND_ERROR :

  // Non-critical error.
  // Tell customer the order could not be completed and to try again later.
  // Notify appropriate internal resources of the error.

break;

// An error occurred while sending the request.

case CYBS_S_SEND_ERROR :

  // Non-critical error.
  // Tell customer the order could not be completed and to try again later.

break;

// An error occurred while waiting for or retrieving the reply.

case CYBS_S_RECEIVE_ERROR :

  // Critial error.
  // Tell customer the order could not be completed and to try again later.
  // Notify appropriate internal resources of the error.
  // See the sample code for more information about handling critical errors.

break;

// An error occurred after receiving and during processing of the reply.

case CYBS_S_POST_RECEIVE_ERROR :

  // Critical error.
  // Tell customer the order could not be completed and to try again later.
  // Look at _raw_reply in pReply for the raw reply.
  // Notify appropriate internal resources of the error.
  // See the sample code for more information about handling critical errors.
break;

// CriticalServerError fault

case CYBS_S_CRITICAL_SERVER_FAULT :

  // Critial error.
  // Tell customer the order could not be completed and to try again later.
  // Read the various fault details from the pReply.
  // Notify appropriate internal resources of the fault.
  // See the sample code for more information about reading fault details and
  // handling a critical error.

break;
```

```
    // ServerError fault

    case CYBS_S_SERVER_FAULT :

      // Non-critical error.
      // Tell customer the order could not be completed and to try again later.
      // Read the various fault details from the pReply.
      // See the sample code for information about reading fault details.

    break;

    // Other fault

    case CYBS_S_OTHER_FAULT :

      // Non-critical error.
      // Tell customer the order could not be completed and to try again later.
      // Read the various fault details from pReply.
      // Notify appropriate internal resources of the fault.
      // See the sample code for information about reading fault details.

    break;

    // HTTP error

    case CYBS_S_HTTP_ERROR :

      // Non-critical error.
      // Tell customer the order could not be completed and to try again later.
      // Look at _raw_reply in pReply for the raw reply.

      break;

    default :

      // Unknown error

  }
}
```

Processing the Reason Codes

After the CyberSource server processes your request, it sends a reply message that contains information about the services you requested. You receive different fields depending on the services you request and the outcome of each service.

To use the reply information, you must integrate it into your system and any other system that uses that data. For example, you can store the reply information in a database and send it to other back office applications.

You must write an error handler to process the reply information that you receive from CyberSource. Do not show the reply information directly to customers. Instead, present an appropriate response that tells customers the result.

Important Because CyberSource may add reply fields and reason codes at any time, you should parse the reply data according to the names of the fields instead of their order in the reply.

The most important reply fields to evaluate are the following:

- **decision**: A one-word description of the results of your request. The decision is one of the following:
  - `ACCEPT` if the request succeeded
  - `REJECT` if one or more of the services in the request was declined
  - `REVIEW` if you are using CyberSource Decision Manager and it flags the order for review. See "Handling Decision Manager Reviews," page 98 for more information.
  - `ERROR` if there was a system error. See "Retrying When System Errors Occur," page 99 for more information.
- **reasonCode**: A numeric code that provides more specific information about the results of your request.

You also receive a reason code for each service in your request. You can use these reason codes to determine whether a specific service succeeded or failed. If a service fails, other services in your request may not run. For example, if you request a credit card authorization and capture, and the authorization fails, the capture does not run. The reason codes for each service are described in the *Credit Card Services User Guide* for CyberSource Essentials merchants or in the service developer guide for CyberSource Advanced merchants.

Important CyberSource reserves the right to add new reason codes at any time. If your error handler receives a reason code that it does not recognize, it should use the decision to interpret the reply.

```
// Example of how to handle reason codes
  // Success

  if( reason == "100" ) {
    printf(
      "Request ID: %s\nAuthorizedAmount: %s\nAuthorization Code: %s\n",
      cybs_get(pReply, "requestID"),
      cybs_get(pReply, "ccAuthReply_amount"),
      cybs_get(pReply, "ccAuthReply_authorizationCode") );
  }

  // Insufficient funds

  else if (reason == "204") {
    printf(
    "Insufficient funds in account. Please use a different
     card or select another form of payment." ) ;
  }

    // add other reason codes here that you must handle specifically

  else {

    // For all other reason codes, return NULL, in which case, you should display
    // a generic message appropriate to the decision value you received.
  }
```

Handling Decision Manager Reviews

If you use CyberSource Decision Manager, you may also receive the `REVIEW` value in the **decision** field. `REVIEW` means that Decision Manager has marked the order for review based on how you configured the Decision Manager rules.

If you will be using Decision Manager, you have to determine how to handle the new `REVIEW` value. Ideally, you will update your order management system to recognize the `REVIEW` response and handle it according to your business rules. If you cannot update your system to handle the `REVIEW` response, CyberSource recommends that you choose one of these options:

- If you authorize and capture the credit card payment at the same time, treat the `REVIEW` response like a `REJECT` response. Rejecting any orders that are marked for review may be appropriate if your product is a software download or access to a Web site. If supported by your processor, you may also want to reverse the authorization.

- If you approve the order after reviewing it, convert the order status to `ACCEPT` in your order management system. You can request the credit card capture without requesting a new authorization.

- If you approve the order after reviewing it but cannot convert the order status to `ACCEPT` in your system, request a new authorization for the order. When processing this new authorization, you must disable Decision Manager. Otherwise the order will be marked for review again. For details about the API field that disables Decision Manager, see the *Decision Manager Developer Guide Using the Simple Order API* (PDF | HTML) or the *Decision Manager Developer Guide Using the SCMP Order API* (PDF | HTML).

Alternately, you can specify a custom business rule in Decision Manager so that authorizations originating from a particular internal IP address at your company are automatically accepted.

If supported by your processor, you may want to reverse the original authorization.

Requesting Multiple Services

When you request multiple services in one request, CyberSource processes the services in a specific order. If a service fails, CyberSource does not process the subsequent services in the request.

For example, in the case of a sale (a credit card authorization and a capture requested together), if the authorization service fails, CyberSource will not process the capture service. The reply you receive only includes reply fields for the authorization.

This following additional example applies **to CyberSource Advanced merchants only.**

Many CyberSource services include “ignore” fields that tell CyberSource to ignore the result from the first service when deciding whether to run the subsequent services. In the case of the sale, even though the issuing bank gives you an authorization code, CyberSource might decline the authorization based on the AVS or card verification results. Depending on your business needs, you might choose to capture these types of declined authorizations anyway. You can set the **businessRules_ignoreAVSResult** field to “true” in your combined authorization and capture request:

```
cybs_add( pRequest, "businessRules_ignoreAVSResult", "true" );
```

This tells CyberSource to continue processing the capture even if the AVS result causes CyberSource to decline the authorization. In this case you would then get reply fields for both the authorization and the capture in your reply.

Note

You are charged only for the services that CyberSource performs.

Retrying When System Errors Occur

You must design your transaction management system to include a way to correctly handle CyberSource system errors. Depending on which payment processor is handling the transaction, the error may indicate a valid CyberSource system error, or it may indicate a processor rejection because of some type of invalid data. In either case, CyberSource recommends that you do not design your system to retry sending a transaction many times in the case of a system error.

Instead, CyberSource recommends that you retry sending the request only two or three times with successively longer periods of time between each retry. For example, after the first system error response, wait 30 seconds and then retry sending the request. If you receive the same error a second time, wait one minute before you send the request again. Depending on the situation, you may decide you can retry sending the request after a longer time period. Determine what is most appropriate for your business situation.

If after several retry attempts you are still receiving a system error, it is possible that the error is actually being caused by a processor rejection and not a CyberSource system error. In that case, we suggest that you either:

- Search for the transaction in the Business Center, look at the description of the error on the Transaction Detail page, and call your processor to determine if and why they are rejecting the transaction.

- Contact CyberSource Customer Support to confirm whether your error is truly caused by a CyberSource system issue.

If TSYS Acquiring Solutions is your processor, you may want to follow the first suggestion as there are several common TSYS Acquiring Solutions processor responses that are returned to you as system errors and that only TSYS Acquiring Solutions can address.

Using XML

This section describes how to request CyberSource services using XML.

Requesting CyberSource Services

To request CyberSource services, write code that:

- Collects information for the CyberSource services that you will use
- Assembles the order information into requests
- Sends the requests to the CyberSource server

Important The CyberSource servers do not support persistent HTTP connections.

- Processes the reply information

The instructions in this section explain how to use C/C++ to request CyberSource services. For a list of API fields to use in your requests, see "Related Documents," page 21.

Sample Code

We suggest that you examine the name-value pair sample code provided in `authCaptureSample.c` before implementing your code to process XML requests. The sample will give you a basic understanding of how to request CyberSource services. The sample code file is located in the `<`*`installation directory`*`>/samples/nvp` directory.

After examining that sample code, read this section to understand how to create code to process XML requests. Note that the code in this section's example is incomplete. For a complete sample program, see the `authSample.c` file in the `<`*`installation directory`*`>/samples/xml` directory.

Creating a Request Document

The client allows you to create an XML request document by using any application, then send the request to CyberSource. For example, if you have a customer relationship management (CRM) system that uses XML to communicate with other systems, you can use the CRM system to generate request documents.

The request document must validate against the XML schema for CyberSource transactions. To view the schema, go to
https://ics2wsa.ic3.com/commerce/1.x/transactionProcessor
and look at the `xsd` file for the version of the Simple Order API you are using.

Important

Make sure that the elements in your document appear in the correct order. If they do not, your document will not validate, and your request will fail.

The example that is developed in the following sections shows a basic XML document for requesting CyberSource services. In this example, Jane Smith is buying an item for $29.95.

The XML document in this example is incomplete. For a complete example, see the `auth.xml` document in the `samples/xml` directory.

Creating an Empty Request

Add the XML declaration and the document's root element:

```
<?xml version="1.0" encoding="utf-8"?>
<requestMessage xmlns="urn:schemas-cybersource-com:transaction-data-
1.18">
</requestMessage>
```

When you construct a request, you must indicate the correct namespace for the elements, and the namespace must use the same API version that you specify in the configuration settings file. For example, if `targetAPIVersion=1.18` in the `cybs.ini` file, the namespace must be `urn:schemas-cybersource-com:transaction-data-1.18`.

Note

The XML document that you receive in the reply always uses a prefix of `c:` (for example, `xmlns:c="urn:schemas-cybersource-com:transaction-data-1.18"`). Make sure you use an XML parser that supports namespaces.

Adding the Merchant ID

You next add the CyberSource merchant ID to the request.

Note

If you specify a merchant ID in the XML document, it overrides the merchant ID you specify in the configuration settings file.

```
<requestMessage xmlns="urn:schemas-cybersource-com:transaction-data-
1.18">
   <merchantID>infodev</merchantID>
</requestMessage>
```

Adding Services to the Request

You next indicate the service that you want to use by creating an element for that service in the request, then setting the element's `run` attribute to `true`. For example, to request a credit card authorization:

```
<requestMessage xmlns="urn:schemas-cybersource-com:transaction-data-
1.18">
  <merchantID>infodev</merchantID>
  <ccAuthService run="true"/>
</requestMessage>
```

Requesting a Sale

You can request multiple services by adding additional elements. For example, if you fulfill the order immediately, you can request a credit card authorization and capture together (referred to as a “sale”):

```
<requestMessage xmlns="urn:schemas-cybersource-com:transaction-data-
1.18">
  <merchantID>infodev</merchantID>
  <ccAuthService run="true"/>
  <ccCaptureService run="true"/>
</requestMessage>
```

Adding Service-Specific Fields to the Request

You next add the fields that are used by the services you are requesting. Most fields are child elements of container elements; for example, a `<card>` element contains the customer’s credit card information.

```
<requestMessage xmlns="urn:schemas-cybersource-com:transaction-data-
1.18">
  <merchantID>infodev</merchantID>
  <billTo>
    <firstName>Jane</firstName>
    <lastName>Smith</lastName>
  </billTo>
  <item id="0">
    <unitPrice>29.95</unitPrice>
  </item>
  <card>
    <accountNumber>4111111111111111</accountNumber>
  </card>
  <ccAuthService run="true"/>
</requestMessage>
```

The example above shows only a partial list of the fields you must send. Refer to "Requesting CyberSource Services," page 90 for information about the guides that list all of the fields for the services that you are requesting.

Sending Requests

Once you have created an XML document, you use C/C++ to send the request to CyberSource.

Adding the Use Statement

First add the include statement for the `cybersource.h` file:

```
#include "cybersource.h"
```

Loading the Configuration Settings

Next use cybs_load_config() to create a new CybsMap structure and load the configuration settings from a file:

```
const char CYBS_CONFIG_INI_FILE[] = "../cybs.ini";
pConfig = cybs_load_config( CYBS_CONFIG_INI_FILE );
```

You could instead create an empty CybsMap structure and add each configuration setting separately. You could also use a combination of the two methods: You could read the settings from a file and then add new settings using the cybs_add() function to override the settings read from the file.

Note

The namespace that you specify in the XML document must use the same API version that you specify in the configuration settings file. For example, if `targetAPIVersion=1.18` in the file, the namespace must be `urn:schemas-cybersource-com:transaction-data-1.18`. The example code below retrieves the API version from the configuration settings file and places it in the XML document.

Creating the Empty Request and Reply

Next use cybs_create_map() to create the request and reply:

```
pRequest = cybs_create_map();
pReply = cybs_create_map();
```

Reading the XML Document

Next, read the XML document and add the information to the request.

```
const char CYBS_XML_INPUT_FILE[] = "./myXMLDocument.xml";

// Read the XML document and store in a variable called szXML.
// See the authSample.c sample code for instructions on reading the
// XML document.

// Add the XML document to the request.

cybs_add( pRequest, CYBS_SK_XML_DOCUMENT, szXML );
```

Sending the Request

You next send the request:

```
status = cybs_run_transaction( pConfig, pRequest, pReply );
```

Interpreting Replies

Handling the Return Status

The `status` value is the handle returned by the cybs_run_transaction() function. The `status` indicates whether the CyberSource server received the request, the client received the reply, or there were any errors or faults during transmission. See "Possible Return Status Values," page 86 for descriptions of each status value. For a different example, see the `authSample.c` file in the client's *<installation directory>*/`xmlSample` directory.

```
if( status == CYBS_S_OK ) {

  // Read the value of the "decision" in pReply.

  decision = cybs_get( pReply, "decision" );

  // If decision=ACCEPT, indicate to the customer that the request was successful.
  // If decision=REJECT, indicate to the customer that the order was not approved.
  // If decision=ERROR, indicate to the customer that there was an error and to try
  // again later.

  // Now get reason code results:

  reason = cybs_get( pReply, "reasonCode" );

  // See "Processing the Reason Codes," page 96 for how to process the
  // reasonCode from the reply.

} else {

  handleError( status, pRequest, pReply );

}

//---------------------
void handleError( CybsStatus stat, CybsMap* preq, CybsMap* prpl )
//---------------------

{

  // handleError shows how to handle the different errors that can occur.

  const char* pstr;
  pstr = cybs_get( prpl, CYBS_SK_ERROR_INFO );
  switch( stat ) {

    // An error occurred before the request could be sent.

    case CYBS_S_PRE_SEND_ERROR :

      // Non-critical error.
      // Tell customer the order could not be completed and to try again later.
      // Notify appropriate internal resources of the error.

    break;

    // An error occurred while sending the request.

    case CYBS_S_SEND_ERROR :

      // Non-critical error.
      // Tell customer the order could not be completed and to try again later.
    break;

    // An error occurred while waiting for or retrieving the reply.

    case CYBS_S_RECEIVE_ERROR :

      // Critial error.
      // Tell customer the order could not be completed and to try again later.
      // Notify appropriate internal resources of the error.
      // See the sample code for more information about handling critical errors.

    break;

    // An error occurred after receiving and during processing of the reply.

    case CYBS_S_POST_RECEIVE_ERROR :

      // Critical error.
      // Tell customer the order could not be completed and to try again later.
      // Look at _raw_reply in pReply for the raw reply.

      // Notify appropriate internal resources of the error.
      // See the sample code for more information about handling critical errors.

    break;

    // CriticalServerError fault

    case CYBS_S_CRITICAL_SERVER_FAULT :

      // Critial error.
      // Tell customer the order could not be completed and to try again later.
      // Read the various fault details from the pReply.
      // Notify appropriate internal resources of the fault.

    // ServerError fault

    case CYBS_S_SERVER_FAULT :

      // Non-critical error.
      // Tell customer the order could not be completed and to try again later.
      // Read the various fault details from pReply.
      // See the sample code for information about reading fault details.

   break;

    // Other fault

    case CYBS_S_OTHER_FAULT :
      // Non-critical error.
      // Tell customer the order could not be completed and to try again later.
      // Read the various fault details from pReply.
      // Notify appropriate internal resources of the fault.
      // See the sample code for information about reading fault details.

    break;

    // HTTP error

    case CYBS_S_HTTP_ERROR :

      // Non-critical error.
      // Tell customer the order could not be completed and to try again later.
      // Look at _raw_reply in pReply for the raw reply.

      break;
    default :

      // Unknown error

  }
}
```

Processing the Reason Codes

After the CyberSource server processes your request, it sends a reply message that contains information about the services you requested. You receive different fields depending on the services you request and the outcome of each service.

To use the reply information, you must integrate it into your system and any other system that uses that data. For example, you can store the reply information in a database and send it to other back office applications.

You must write an error handler to process the reply information that you receive from CyberSource. Do not show the reply information directly to customers. Instead, present an appropriate response that tells customers the result.

Important Because CyberSource may add reply fields and reason codes at any time, you should parse the reply data according to the names of the fields instead of their order in the reply.

The most important reply fields to evaluate are the following:

- **decision**: A one-word description of the results of your request. The decision is one of the following:
    - `ACCEPT` if the request succeeded
    - `REJECT` if one or more of the services in the request was declined
    - `REVIEW` if you use CyberSource Decision Manager and it flags the order for review. See "Handling Decision Manager Reviews," page 111 for more information.
    - `ERROR` if there was a system error. See "Retrying When System Errors Occur," page 113 for more information.
- **reasonCode**: A numeric code that provides more specific information about the results of your request.

You also receive a reason code for each service in your request. You can use these reason codes to determine whether a specific service succeeded or failed. If a service fails, other services in your request may not run. For example, if you request a credit card authorization and capture, and the authorization fails, the capture does not run. The reason codes for each service are described in the *Credit Card Services User Guide* for CyberSource Essentials merchants or in the service developer guide for CyberSource Advanced merchants.

Important CyberSource reserves the right to add new reason codes at any time. If your error handler receives a reason code that it does not recognize, it should use the decision to interpret the reply.

```
// Example of how to handle reason codes

  // Success

  if( reason == "100" ) {
    printf(
      "Request ID: %s\nAuthorizedAmount: %s\nAuthorization Code: %s\n",
      cybs_get(pReply, "requestID"),
      cybs_get(pReply, "ccAuthReply_amount"),
      cybs_get(pReply, "ccAuthReply_authorizationCode") );
  }

  // Insufficient funds

  else if (reason == "204") {
    printf(
    "Insufficient funds in account. Please use a different
     card or select another form of payment." ) ;
  }

    // add other reason codes here that you must handle specifically

  else {

    // For all other reason codes, return NULL, in which case, you should display a
    // generic message appropriate to the decision value you received.

  }
```

Handling Decision Manager Reviews

If you use CyberSource Decision Manager, you may also receive the `REVIEW` value in the **decision** field. `REVIEW` means that Decision Manager has marked the order for review based on how you configured the Decision Manager rules.

If you will be using Decision Manager, you have to determine how to handle the new `REVIEW` value. Ideally, you will update your order management system to recognize the `REVIEW` response and handle it according to your business rules. If you cannot update your system to handle the `REVIEW` response, CyberSource recommends that you choose one of these options:

- If you authorize and capture the credit card payment at the same time, treat the `REVIEW` response like a `REJECT` response. Rejecting any orders that are marked for review may be appropriate if your product is a software download or access to a Web site. If supported by your processor, you may also want to reverse the authorization.

- If you approve the order after reviewing it, convert the order status to `ACCEPT` in your order management system. You can request the credit card capture without requesting a new authorization.

- If you approve the order after reviewing it but cannot convert the order status to `ACCEPT` in your system, request a new authorization for the order. When processing this new authorization, you must disable Decision Manager. Otherwise the order will be marked for review again. For details about the API field that disables Decision Manager, see the *Decision Manager Developer Guide Using the Simple Order API* (PDF | HTML) or the *Decision Manager Developer Guide Using the SCMP Order API* (PDF | HTML).

Alternately, you can specify a custom business rule in Decision Manager so that authorizations originating from a particular internal IP address at your company are automatically accepted.

If supported by your processor, you may want to reverse the original authorization.

Requesting Multiple Services

When you request multiple services in one request, CyberSource processes the services in a specific order. If a service fails, CyberSource does not process the subsequent services in the request.

For example, in the case of a sale (a credit card authorization and a capture requested together), if the authorization service fails, CyberSource will not process the capture service. The reply you receive only includes reply fields for the authorization.

This following additional example applies **to CyberSource Advanced merchants only.**

Many CyberSource services include “ignore” fields that tell CyberSource to ignore the result from the first service when deciding whether to run the subsequent services. In the case of the sale, even though the issuing bank gives you an authorization code, CyberSource might decline the authorization based on the AVS or card verification results. Depending on your business needs, you might choose to capture these types of declined authorizations anyway. You can set the **businessRules_ignoreAVSResult** field to “true” in your combined authorization and capture request:

```
<businessRules>
    <ignoreAVSResult>true</ignoreAVSResult>
</businessRules>
```

This tells CyberSource to continue processing the capture even if the AVS result causes CyberSource to decline the authorization. In this case you would then get reply fields for both the authorization and the capture in your reply.

Note

You are charged only for the services that CyberSource performs.

Retrying When System Errors Occur

You must design your transaction management system to include a way to correctly handle CyberSource system errors. Depending on which payment processor is handling the transaction, the error may indicate a valid CyberSource system error, or it may indicate a processor rejection because of some type of invalid data. In either case, CyberSource recommends that you do not design your system to retry sending a transaction many times in the case of a system error.

Instead, CyberSource recommends that you retry sending the request only two or three times with successively longer periods of time between each retry. For example, after the first system error response, wait 30 seconds and then retry sending the request. If you receive the same error a second time, wait one minute before you send the request again. Depending on the situation, you may decide you can retry sending the request after a longer time period. Determine what is most appropriate for your business situation.

If after several retry attempts you are still receiving a system error, it is possible that the error is actually being caused by a processor rejection and not a CyberSource system error. In that case, we suggest that you either:

- Search for the transaction in the Business Center, look at the description of the error on the Transaction Detail page, and call your processor to determine if and why they are rejecting the transaction.
- Contact CyberSource Customer Support to confirm whether your error is truly caused by a CyberSource system issue.

If TSYS Acquiring Solutions is your processor, you may want to follow the first suggestion as there are several common TSYS Acquiring Solutions processor responses that are returned to you as system errors and that only TSYS Acquiring Solutions can address.

Advanced Configuration Information

Using Alternate Server Configuration Settings

You use the serverURL and namespaceURI configuration settings if CyberSource changes the convention we use to specify the server URL and namespace URI, but we have not had the opportunity to update the client yet.

For example, these are the server URLs and namespace URI for accessing the CyberSource services using the Simple Order API version 1.18:

- Test server URL:

  ```
  https://ics2wstesta.ic3.com/commerce/1.x/transactionProcessor
  ```

- Production server URL:

  ```
  https://ics2wsa.ic3.com/commerce/1.x/transactionProcessor
  ```

- Namespace URI:

  ```
  urn:schemas-cybersource-com:transaction-data-1.18.
  ```

Note

If view the above URLs in a web browser, a list of the supported API versions and the associated schema files are displayed.

If in the future CyberSource changes these conventions, but does not provide a new version of the client, you can configure your existing client to use the new server and namespace conventions required by the CyberSource server.

Configuring for Multiple Merchant IDs

If you have multiple merchant IDs, or if you are a reseller handling multiple merchants, you can have different configuration settings for different merchant IDs. You set these in the configuration object that you pass to the cybs_run_transaction() function. When using the samples provided in the client package, you set the configuration parameters in `cybs.ini` file.

All of the properties except merchantID can be prefixed with "`<merchantID>.`" to specify the settings for a specific merchant.

Example Merchant-Specific Properties Settings

If you have a merchant with merchant ID of `merchant123`, and you want enable logging only for that merchant, you can set the enableLog parameter to `true` for all requests that have `merchant123` as the merchant ID:

```
merchant123.enableLog=true
enableLog=false
```

The client disables logging for all other merchants.

CHAPTER

.NET 1.1 Client

4

Important

- The .NET 1.1 client for the Simple Order API is supported on 32-bit operating systems only.
- If you are building an application to sell to others, see Appendix A, "Using the Client Application Fields," on page 390. This appendix has a list of API fields you can use in your request that describe the application, its version, and its user. If you use these fields in your request, you can view their values in the Transaction Search Details window of the Business Center.

Choosing an API Variation

With this client package, you can use any of the three variations of the Simple Order API:

- Name-value pairs, which are simpler to use than XML
- XML, which requires you to create and parse XML documents
- SOAP (Simple Object Access Protocol) 1.1, which provides an object-oriented interface

The test that you run immediately after installing the client uses name-value pairs.

A Note about the API and Client Versions

CyberSource updates the Simple Order API on a regular basis to introduce new API fields and functionality. To identify the latest version of the API, go to https://ics2wsa.ic3.com/commerce/1.x/transactionProcessor.

This represents the version of the server-side code for the CyberSource services.

If a new version of the API has been released but we have not yet updated the .NET client to use this new version, you can manually update the client to use a different version. For example, when a new version of the API is released, you can update your client to use this new API version. See "Updating the Client to Use a Later API Version," page 128.

Basic C# Program Example

The example below shows the primary code required to send a SOAP request for credit card authorization and process the reply. See "Using SOAP," page 152 for more information.

```
using CyberSource.Soap;
using CyberSource.Soap.CyberSourceWS;
using System;
using System.Configuration;
using System.Net;
using System.Web.Services.Protocols;
namespace Sample {
  class Sample {
    static void Main(string[] args) {
      RequestMessage request = new RequestMessage();
      request.merchantID = "infodev";

      // we want to do Credit Card Authorization in this sample
      request.ccAuthService = new CCAuthService();
      request.ccAuthService.run = "true";

      // add required fields
      request.merchantReferenceCode = "148705832705344";
      BillTo billTo = new BillTo();
      billTo.firstName = "Jane";
      billTo.lastName = "Smith";
      billTo.street1 = "1295 Charleston Road";
      billTo.city = "Mountain View";
      billTo.state = "CA";
      billTo.postalCode = "94043";
      billTo.country = "US";
      billTo.email = "jsmith@example.com";
      request.billTo = billTo;
      Card card = new Card();
      card.accountNumber = "4111111111111111";
      card.expirationMonth = "12";
      card.expirationYear = "2010";
      request.card = card;
      PurchaseTotals purchaseTotals = new PurchaseTotals();
      purchaseTotals.currency = "USD";
      request.purchaseTotals = purchaseTotals;

      // there is one item in this sample
      request.item = new Item[1];
      Item item = new Item();
      item.id = "0";
      item.unitPrice = "29.95";
      request.item[0] = item;
      // See "Interpreting the Reply," page 156 for details about
      // processing the reply for a SOAP transaction.
      try {
        ReplyMessage reply = Client.RunTransaction( request );
      } catch (SignException se) {
        Console.WriteLine( se.ToString() );
      } catch (SoapHeaderException she) {
        Console.WriteLine( she.ToString() );
      } catch (SoapBodyException sbe) {
        Console.WriteLine( sbe.ToString() );
      } catch (WebException we) {
        Console.WriteLine( we.ToString() );
      }
    }
  }
}
```

Installing and Testing the Client

Minimum System Requirements

- Microsoft Windows 2000 or later
- .NET Framework 1.1 or later
- Microsoft Web Services Enhancements (WSE) 2.0 Service Pack 3 or later (name-value pair and SOAP clients only)

Important

Failure to configure your client API host to a unique, public IP address will cause inconsistent transaction results.

The client API request ID algorithm uses a combination of IP address and system time, along with other values. In some architectures this combination might not yield unique identifiers.

Transaction Security Keys

The first thing you must do is create your security key. The client uses the security key to add a digital signature to every request that you send. This signature helps ensure that no one else can use your CyberSource account to process orders. You specify the location of your key when you configure the client.

Important You must generate two transaction security keys—one for the CyberSource production environment and one for the test environment. For information about generating and using security keys, see *Creating and Using Security Keys* (PDF | HTML).

Warning You must protect your security key to ensure that your CyberSource account is not compromised.

Installing the Client

The Simple Order API for .NET Setup Wizard installs and registers files for the client. By default, the wizard installs the name-value pair, XML, and SOAP client variations. If you prefer, you can install only the variations that you plan to use.

To install the client for the first time:

Step 1 Go to the client downloads page on the Support Center and download the latest version of the client.

Step 2 Run the downloaded file.

The Simple Order API for .NET Setup Wizard appears.

Step 3 Follow the instructions in the wizard to install the client.

If you use the default installation directory, the client is installed in
`c:\simapi-net-1.1-n.n.n`
where `n.n.n` is the client version.

Step 4 Test the client. See "Using the Test Applications," page 122.

You have installed and tested the client. You are ready to create your own code for requesting CyberSource services. Finish reading this section, and then move on to:

- "Using Name-Value Pairs," page 129 if you plan to use name-value pairs
- "Using XML," page 140 if you plan to use XML
- "Using SOAP," page 152 if you plan to use SOAP

Upgrading from a Previous Version

To upgrade your client from a previous version:

Step 1 Follow the above installation instructions.

Step 2 In `VS.NET`, remove the reference to the previous client.

Step 3 Add a reference to the new client (`NVPClient.dll`, `SoapClient.dll`, or `XmlClient.dll`).

Step 4 In your code, if you were using the Basic client, replace the references to CyberSource.Basic with CyberSource.NVP.

Step 5 If you are using the XML client, replace any references to the Client.CYBS_NAMESPACE with XmlClient.GetRequestNamespace().

Step 6 Although the previous configuration setting keys are still supported, it is recommended that you now prefix them with "cybs." to prevent them from clashing with any other settings that you may have.

Step 7 CyberSource recommends that you stop hard coding the URL. In other words, replace the key "cybersourceURL" with "cybs.sendToProduction" and set it to true or false (or 0 or 1) as appropriate. The URL will be derived internally from this flag.

You have successfully upgraded your client to the new version.

Using the Test Applications

Each type of client variation—name-value pair, XML, and SOAP—includes two precompiled test applications. You can use these test applications to ensure that the client was installed correctly.

One application requests a single CyberSource service, a credit card authorization. The other requests multiple CyberSource services.

The test applications and their source code are installed with the other files for the client. The following table shows the filename for each test application.

Table 32 File Names of Test Applications

Type of Client	Filename
Name-Value Pair	Single service: ▪ `NVP\Prebuilt\Single.exe` Multiple services: ▪ `NVP\Prebuilt\Multi.exe` Source code: ▪ `NVP\SingleServiceSample\` ▪ `NVP\MultiServiceSample\`
XML	Single service: ▪ `Xml\Prebuilt\Single.exe` Multiple services: ▪ `Xml\Prebuilt\Multi.exe` Source code: ▪ `Xml\SingleServiceSample\` ▪ `Xml\MultiServiceSample\`
SOAP	Single service: ▪ `Soap\Prebuilt\Single.exe` Multiple services: ▪ `Soap\Prebuilt\Multi.exe` Source code: ▪ `Soap\SingleServiceSample\` ▪ `Soap\MultiServiceSample\`

Configuring the Test Applications

Before you run a test application, you must edit its application settings file. The following table describes all of the configuration fields that you can use in this file.

Table 33 Fields in the Settings File

Field Name	Description	Required/ Optional
cybs.connectionLimit	Maximum number of allowed concurrent connections between the client and CyberSource server. For more information on this field and alternate ways to set the connection limits, see "Setting the Connection Limit," page 162.	Optional
cybs.keysDirectory	Directory that contains the pkcs12 security key file, for example: `c:\keys\`	Required
cybs.merchantID	Your CyberSource merchant ID. You can override this value by providing the merchantID field in the request itself.	Optional
cybs. sendToProduction	Flag that indicates whether the transactions for this merchant should be sent to the production server. Use one of these values: ▪ `false`: Do not send to the production server; send to the test server (default setting). ▪ `true`: Send to the production server. **Note** Make sure that if your merchant ID is configured to use the test mode, you send requests to the test server.	Required
cybs.keyFilename	Name of the security key file name for the merchant in the format `<`***`security_key_filename`***`>.p12`.	Optional
cybs.serverURL	Alternate server URL to use. For more information, see "Configuring Your Settings for Multiple Merchants," page 125. Give the complete URL because it will be used exactly as you specify.	Optional

Table 33 Fields in the Settings File (Continued)

Field Name	Description	Required/ Optional
cybs.enableLog	Flag directing the client to log transactions and errors. Possible values: - `false`: Do not enable logging (default setting). - `true`: Enable logging. **Important** Logging can cause very large log files to accumulate. Therefore, CyberSource recommends that you use logging only when troubleshooting problems. To comply with all Payment Card Industry (PCI) and Payment Application (PA) Data Security Standards regarding the storage of credit card and card verification number data, the logs that are generated contain only masked credit card and card verification number data (CVV, CVC2, CVV2, CID, CVN). Follow these guidelines: - Use debugging temporarily for diagnostic purposes only. - If possible, use debugging only with test credit card numbers. - Never store clear text card verification numbers. - Delete the log files as soon as you no longer need them. - Never send email to CyberSource containing personal and account information, such as customers' names, addresses, card or check account numbers, and card verification numbers. For more information about PCI and PABP requirements, see www.visa.com/cisp.	Optional
cybs.logDirectory	Directory to which to write the log file. Note that the client will not create this directory for you; you must specify an existing directory. The client includes a `logs` directory that you can use. Include the path, for example: `c:\simapi-net-2.0.0\logs`.	Required if **cybs. enableLog** is true
cybs.logFilename	Name of the log file. The client uses `cybs.log` by default.	Optional
cybs.logMaximumSize	Maximum size in megabytes for the log file. The default value is `10`. When the log file reaches the specified size, it is archived into `cybs.log.`***<yyyymmdd*****`T`*****hhmmssxxx>*** and a new log file is started. The ***xxx*** indicates milliseconds.	Optional
cybs.timeout	Length of time-out in seconds. The default is 130.	Optional
cybs.proxyURL	URL of a proxy server, for example: `https://proxy.example.com:4909`	Optional
cybs.proxyUser	User name for the proxy server.	Optional
cybs.proxyPassword	Password for the proxy server.	Optional

To test applications:

Step 1 Decide which test application you want to run.

For example, you could decide to run `Single.exe`.

Step 2 Using a text editor, open the settings file for the test application.

The settings file has the same name as the test application, with the extension `config` appended to the name (for example, `Single.exe.config`).

Step 3 Find the `cybs.merchantID` field and change its value to your CyberSource merchant ID.

For example, if your merchant ID is `widgetsinc`:

```
<add key="cybs.merchantID" value="widgetsinc"/>
```

Step 4 Find the `cybs.keysDirectory` field and change its value to the directory that contains your security key.

For example, if your key is in `c:\keys\`:

```
<add key="cybs.keysDirectory" value="c:\keys\"/>
```

Step 5 Edit other fields as necessary. See Table 33, page 123 for a complete list.

Step 6 Save and close the settings file.

Configuring Your Settings for Multiple Merchants

If you have multiple merchant IDs, or if you are a reseller handling multiple merchants, you can configure the settings to allow different configurations for different merchant IDs.

To specify the settings for a specific merchant, prefix all settings, except for `cybs.merchantID` and the `cybs.proxy*`, with `<merchantID>.`

The `cybs.proxy*` wildcard refers to the `proxyURL`, `proxyUser`, `proxyPassword` settings.

Example **You have a new merchant with merchant ID of `NewMerchant`. To send only test transactions for this merchant, you can set all requests for `NewMerchant` to go to the test server:**

```
<add key="cybs.NewMerchant.sendToProduction" value="false"/>
<add key="cybs.sendToProduction" value="true"/>
```

With the second line of the example, the client will send all other requests to the production server.

Running the Test Applications

To run test applications:

Step 1 Open a Windows command-line shell.

Step 2 Change to the directory where the test application is located.

Step 3 Type the name of the test application, then press Enter.

The test application requests an CyberSource service, interprets the reply, and prints information about the result. If you receive a .NET exception, use the error message to debug the problem.

Deploying the Client to Another Computer

To deploy the client to another computer without running the installer provided by CyberSource, you must include all the files from the `lib` directory in your custom installer.

Then, you must register the `CybsWSSecurity.dll` either in your installation script or on the command prompt.

To register CybsWSSecurity.dll:

Step 1 Open a command prompt.

Step 2 Go to the directory where you installed the client files.

Step 3 Type the following:

```
regsvr32 CybsWSSecurity.dll
```

The client is now ready to be used on the machine.

Going Live

When you complete all of your system testing and are ready to accept real transactions from your customers, your deployment is ready to *go live*.

Note

After your deployment goes live, use real card numbers and other data to test every card type you support. Because these are real transactions in which you are buying from yourself, use small monetary amounts to do the tests. Process an authorization, then capture the authorization, and later refund the money. Use your bank statements to verify that money is deposited into and withdrawn from your merchant bank account as expected. If you have more than one CyberSource merchant ID, test each one separately.

CyberSource Essentials Merchants

If you use CyberSource Essentials services, you can use the Business Center site to go live. For a description of the process of going live, see the "Steps for Getting Started" section in *Getting Started with CyberSource Essentials*.

Important

You must also configure your client so that it sends transactions to the production server and not the test server. See the description of the configuration field "cybs. sendToProduction," page 123.

CyberSource Advanced Merchants

If you use CyberSource Advanced services, see the "Steps for Getting Started" chapter in *Getting Started with CyberSource Advanced* for information about going live.

When your deployment goes live, your CyberSource account is updated so that you can send transactions to the CyberSource production server. If you have not already done so, you must provide your banking information to CyberSource so that your processor can deposit funds to your merchant bank account.

After CyberSource confirms that your account is live, make sure that you update your system so that it can send requests to the production server (`ics2wsa.ic3.com`) using your security key for the production environment. The test server (`ics2wstesta.ic3.com`) cannot be used for real transactions. For more information about sending transactions to the production server, see the description of the configuration field "cybs. sendToProduction," page 123.

Updating the Client to Use a Later API Version

CyberSource periodically updates the Simple Order API. You can update your existing client to work with the new API version. Go to https://ics2wsa.ic3.com/commerce/1.x/transactionProcessor for a list of the available API versions.

Alternately, if a new client is available that works with the later API version, you can download that new client.

Note The new client may have new functionality unrelated to the changes in the API. Read the release notes in the `CHANGES` file to determine if the new client contains new functionality that you want to use.

Name-Value Pair and SOAP Client Variations

To update the client to use a new API when using name-value pair and SOAP client variations:

Step 1 Depending on whether you use name-value pairs or SOAP, load `nvp11.sln` or `soap11.sln` into Visual Studio.NET.

Step 2 Inside Visual Studio.NET, enter the URL of the new WSDL file in the "Web Reference URL" of the "CyberSourceWS" web reference.

This generates a new `Reference.cs` in the `Web References\CyberSourceWS` directory.

Step 3 Change the base class in `Reference.cs` from

```
System.Web.Services.Protocols.SoapHttpClientProtocol
```

to

```
Microsoft.Web.Services2.WebServicesClientProtocol
```

Step 4 Update the constructor so that it accepts the parameter `"string url"` and replace the hard-coded URL assigned to `"this.URL"` with this parameter. For example:

```
// for nvp client
public NVPTransactionProcessor(string url) {
this.Url = url;
}

// for soap client
public TransactionProcessor(string url) {
this.Url = url;
}
```

Step 5 Build the solution.

Your client can use the new version of the API.

XML Client

Updating the client is unnecessary. Start using the new namespace URI in your input XML documents. The client automatically uses the specified version.

Using Name-Value Pairs

This section explains how to request CyberSource services by using name-value pairs.

Requesting CyberSource Services

To request CyberSource services, write code that:

- Collects information for the CyberSource services that you will use
- Assembles the order information into requests
- Sends the requests to the CyberSource server

Important The CyberSource servers do not support persistent HTTP connections.

- Processes the reply information

The instructions in this section explain how to write C# programs that request CyberSource services. For a list of API fields to use in your requests, see "Related Documents," page 21.

Sample Code

The code in this section's examples is incomplete. For complete sample programs, see the source code in the client's `NVP\SingleServiceSample` and `NVP\MultiServiceSample` directories.

Creating and Sending the Request

To use any CyberSource service, you must create and send a request that includes the required information for that service.

The example developed in the following sections shows basic code for requesting CyberSource services. In this example, Jane Smith is buying an item for $29.95.

Creating a New Visual Studio .NET Project

To get started, create a new project in Visual Studio .NET, and add a reference to the client library, `NVPClient.dll`, which is located in the client installation directory. You must also add a reference to the library `System.Web.Services.dll`, which is part of the .NET Framework.

Importing the Client Classes

In the code for your application, add the following import statements:

```
using CyberSource.NameValuePair;
using System;
using System.Collections;
using System.Configuration;
using System.Net;
using System.Web.Services.Protocols;
```

Creating an Empty Request

You next create a hashtable that holds the request fields:

```
Hashtable request = new Hashtable();
```

Adding the Merchant ID

You next optionally add your CyberSource merchant ID to the request:

```
request.Add( "merchantID", "infodev" );
```

This value overrides any value you set with the merchantID configuration setting (see Table 33, page 123).

Adding Services to the Request

You next indicate the service that you want to use by adding a field to the request. For example, to request a credit card authorization:

```
request.Add( "ccAuthService_run", "true" );
```

Requesting a Sale

You can request multiple services by adding additional fields to the request. For example, if you fulfill the order immediately, you can request a credit card authorization and capture together (also referred to as a “sale”):

```
request.Add( "ccAuthService_run", "true" );
request.Add( "ccCaptureService_run", "true" );
```

Adding Service-Specific Fields to the Request

You next add the fields that are used by the services you are requesting. If you request multiple services and they share common fields, you must add the field once only.

```
request.Add( "billTo_firstName", "Jane" );
request.Add( "billTo_lastName", "Smith" );
request.Add( "card_accountNumber", "4111111111111111" );
request.Add( "item_0_unitPrice", "29.95" );
```

The example above shows only a partial list of the fields you must send. Refer to "Related Documents," page 21 for information about the guides that list all of the fields for the services that you are requesting.

Sending the Request

You next send the request to CyberSource, store the reply in a new hashtable, and catch several exceptions that you might receive:

```
try {
  Hashtable reply = Client.RunTransaction( request );
  SaveOrderState();

  // "Using the Decision and Reason Code," page 135 describes the
  // ProcessReply method.
  ProcessReply( reply );
} catch (SignException se) {
  SaveOrderState();
  Console.WriteLine( se.ToString() );
} catch (SoapHeaderException she) {
  SaveOrderState();
  Console.WriteLine( she.ToString() );
} catch (SoapBodyException sbe) {
  SaveOrderState();
  /*
   * Some types of SoapBodyException indicate that the transaction may
   * have been completed by CyberSource. The sample code shows how to
   * identify these exceptions. If you receive such an exception, and
   * your request included a payment service, you should use the
   * CyberSource transaction search screens to determine whether the
   * transaction was processed.
   */
  Console.WriteLine( sbe.ToString() );
} catch (WebException we) {
  SaveOrderState();
  /*
   * Some types of WebException indicate that the transaction may
   * have been completed by CyberSource. The sample code shows how to
   * identify these exceptions. If you receive such an exception, and
   * your request included a payment service, you should use the
   * CyberSource transaction search screens to determine whether the
   * transaction was processed.
   */
  Console.WriteLine( we.ToString() );
}
```

```
private static void SaveOrderState() {
  /*
   * This is where you store the order state in your system for
   * post-transaction analysis. Be sure to store the customer
   * information, the values of the reply fields, and the details
   * of any exceptions that occurred.
   */
}
```

In the previous example, when an exception occurs, the exception is printed to the console. Your web store should also display a message to the customer indicating that you were unable to process the order. The sample code for the name-value pair client shows you how to provide feedback to the customer.

Also, if the transaction fails, and the request did not include any payment services, you may be able to resend the transaction. The sample code for the name-value pair client shows you how to do this.

Interpreting the Reply

After the CyberSource server processes your request, it sends a reply message that contains information about the services you requested. You receive different fields depending on the services you request and the outcome of each service.

To use the reply information, you must integrate it into your system and any other system that uses that data. For example, you can store the reply information in a database and send it to other back office applications.

You must write an error handler to process the reply information that you receive from CyberSource. Do not show the reply information directly to customers. Instead, present an appropriate response that tells customers the result.

Important Because CyberSource may add reply fields and reason codes at any time, you should parse the reply data according to the names of the fields instead of their order in the reply.

The most important reply fields to evaluate are the following:

- **decision**: A one-word description of the results of your request. The decision is one of the following:
  - `ACCEPT` if the request succeeded
  - `REJECT` if one or more of the services in the request was declined
  - `REVIEW` if you use CyberSource Decision Manager and it flags the order for review. See "For CyberSource Advanced Merchants: Handling Decision Manager Reviews," page 137 for more information.
  - `ERROR` if there was a system error. See "Retrying When System Errors Occur," page 139 for important information about handling retries in the case of system errors.
- **reasonCode**: A numeric code that provides more specific information about the results of your request.

You also receive a reason code for each service in your request. You can use these reason codes to determine whether a specific service succeeded or failed. If a service fails, other services in your request may not run. For example, if you request a credit card authorization and capture, and the authorization fails, the capture does not run. The reason codes for each service are described in the *Credit Card Services User Guide* for CyberSource Essentials merchants or in the service developer guide for CyberSource Advanced merchants.

Important CyberSource reserves the right to add new reason codes at any time. If your error handler receives a reason code that it does not recognize, it should use the decision to interpret the reply.

Using the Decision and Reason Code

The following example shows how you can use the decision and the reason code to display an appropriate message to your customer.

```
private static bool ProcessReply( Hashtable reply ) {
  string template = GetTemplate(
    ((string)reply["decision"]).ToUpper() );
  string content  = GetContent( reply );

  // This example writes the message to the console. Choose an
  // appropriate display method for your own application.
  Console.WriteLine( template, content );
}

private static string GetTemplate( string decision ) {
  // Retrieves the text that corresponds to the decision.

  if ("ACCEPT".Equals( decision )) {
    return( "The order succeeded.{0}" );
  }
  if ("REJECT".Equals( decision )) {
    return( "Your order was not approved.{0}" );
  }

  // ERROR, or an unknown decision
  return( "Your order could not be completed at this time.{0}" +
          "\nPlease try again later." );
}
private static string GetContent( Hashtable reply ) {
  /*
   * Uses the reason code to retrieve more details to add to the
   * template.
   *
   * The messages returned in this example are meant to demonstrate
   * how to retrieve the reply fields. Your application should
   * display user-friendly messages.
   */
  int reasonCode = int.Parse( (string) reply["reasonCode"] );
  switch (reasonCode) {

    // Success
    case 100:
      return( "\nRequest ID: " + reply["requestID"] );

    // Missing field or fields
    case 101:
      return( "\nThe following required fields are missing: " +
              EnumerateValues( reply, "missingField" ) );
    // Invalid field or fields
    case 102:
      return( "\nThe following fields are invalid: " +
              EnumerateValues( reply, "invalidField" ) );
    // Insufficient funds
    case 204:
      return( "\nInsufficient funds in the account. Please use a " +
              "different card or select another form of payment." );

    // Add additional reason codes here that you must handle
    // more specifically.
    default:

      // For all other reason codes (for example, unrecognized reason
      // codes, or codes that do not require special handling),
      // return an empty string.
      return( String.Empty );
  }
}
private static string EnumerateValues( Hashtable reply,
                                       string fieldName ) {
  System.Text.StringBuilder sb = new System.Text.StringBuilder();
  string val = "";
  for (int i = 0; val != null; ++i) {
    val = (string) reply[fieldName + "_" + i];
    if (val != null) {
      sb.Append( val + "\n" );
    }
  }

  return( sb.ToString() );
}
```

For CyberSource Advanced Merchants: Handling Decision Manager Reviews

The information in this section applies only to CyberSource Advanced merchants.

If you use CyberSource Decision Manager, you may also receive the `REVIEW` value in the **decision** field. `REVIEW` means that Decision Manager has marked the order for review based on how you configured the Decision Manager rules.

If you will be using Decision Manager, you have to determine how to handle the new `REVIEW` value. Ideally, you will update your order management system to recognize the `REVIEW` response and handle it according to your business rules. If you cannot update your system to handle the `REVIEW` response, CyberSource recommends that you choose one of these options:

- If you authorize and capture the credit card payment at the same time, treat the `REVIEW` response like a `REJECT` response. Rejecting any orders that are marked for review may be appropriate if your product is a software download or access to a Web site. If supported by your processor, you may also want to reverse the authorization.

- If you approve the order after reviewing it, convert the order status to `ACCEPT` in your order management system. You can request the credit card capture without requesting a new authorization.

- If you approve the order after reviewing it but cannot convert the order status to `ACCEPT` in your system, request a new authorization for the order. When processing this new authorization, you must disable Decision Manager. Otherwise the order will be marked for review again. For details about the API field that disables Decision Manager, see the *Decision Manager Developer Guide Using the Simple Order API* (PDF | HTML) or the *Decision Manager Developer Guide Using the SCMP Order API* (PDF | HTML).

Alternately, you can specify a custom business rule in Decision Manager so that authorizations originating from a particular internal IP address at your company are automatically accepted.

If supported by your processor, you may want to reverse the original authorization.

Requesting Multiple Services

When you request multiple services in one request, CyberSource processes the services in a specific order. If a service fails, CyberSource does not process the subsequent services in the request.

For example, in the case of a sale (a credit card authorization and a capture requested together), if the authorization service fails, CyberSource will not process the capture service. The reply you receive only includes reply fields for the authorization.

Many CyberSource services include “ignore” fields that tell CyberSource to ignore the result from the first service when deciding whether to run the subsequent services. In the case of the sale, even though the issuing bank gives you an authorization code, CyberSource might decline the authorization based on the AVS or card verification results. Depending on your business needs, you might choose to capture these types of declined authorizations anyway. You can set the **businessRules_ignoreAVSResult** field to `"true"` in your combined authorization and capture request:

```
request.put( "businessRules_ignoreAVSResult", "true" );
```

This tells CyberSource to continue processing the capture even if the AVS result causes CyberSource to decline the authorization. In this case you would then get reply fields for both the authorization and the capture in your reply.

Note You are charged only for the services that CyberSource performs.

Retrying When System Errors Occur

You must design your transaction management system to include a way to correctly handle CyberSource system errors. Depending on which payment processor is handling the transaction, the error may indicate a valid CyberSource system error, or it may indicate a processor rejection because of some type of invalid data. In either case, CyberSource recommends that you do not design your system to retry sending a transaction many times in the case of a system error.

Instead, CyberSource recommends that you retry sending the request only two or three times with successively longer periods of time between each retry. For example, after the first system error response, wait 30 seconds and then retry sending the request. If you receive the same error a second time, wait one minute before you send the request again. Depending on the situation, you may decide you can retry sending the request after a longer time period. Determine what is most appropriate for your business situation.

If after several retry attempts you are still receiving a system error, it is possible that the error is actually being caused by a processor rejection and not a CyberSource system error. In that case, we suggest that you either:

- Search for the transaction in the Business Center, look at the description of the error on the Transaction Detail page, and call your processor to determine if and why they are rejecting the transaction.

- Contact CyberSource Customer Support to confirm whether your error is truly caused by a CyberSource system issue.

If TSYS Acquiring Solutions is your processor, you may want to follow the first suggestion as there are several common TSYS Acquiring Solutions processor responses that are returned to you as system errors and that only TSYS Acquiring Solutions can address.

Creating an Application Settings File

After you finish writing code for your integration, you must create an application settings file. This file must contain, at a minimum, the following information:

- The directory that contains your security key
- The location of the CyberSource server

See Table 33, "Fields in the Settings File" for a complete list of settings.

You can use the settings files that come with the sample applications as a starting point for your own settings file. See "Configuring the Test Applications," page 123 for more information.

Using XML

This section explains how to request CyberSource services by using XML.

Requesting CyberSource Services

To request CyberSource services, write code that:

- Collects information for the CyberSource services that you will use
- Assembles the order information into requests
- Sends the requests to the CyberSource server
- Processes the reply information

Important The CyberSource servers do not support persistent HTTP connections.

The instructions in this section explain how to write C# programs that request CyberSource services. For a list of API fields to use in your requests, see "Related Documents," page 21.

Creating a Request Document

The XML client allows you to create an XML request document using any application, then send the request to CyberSource. For example, if you have a customer relationship management (CRM) system that uses XML to communicate with other systems, you can use the CRM system to generate request documents.

The request document must validate against the XML schema for CyberSource transactions. To view the schema, go to
https://ics2wsa.ic3.com/commerce/1.x/transactionProcessor
and look at the `xsd` file for the version of the Simple Order API you are using.

Important Make sure that the elements in your document appear in the correct order. If they do not, your document will not validate, and your request will fail.

The following example shows a basic XML document for requesting CyberSource services. In this example, Jane Smith is buying an item for $29.95.

Note The XML document in this example is incomplete. For complete examples, see the documents in the `Xml\SingleServiceSample\` and `Xml\MultiServiceSample\` directories.

Creating an Empty Request

Add the XML declaration and the document's root element:

```
<?xml version="1.0" encoding="utf-8"?>
<requestMessage xmlns="urn:schemas-cybersource-com:transaction-data-
1.17">
</requestMessage>
```

When you construct a request, you must indicate the correct namespace for the elements, and the namespace must use the same API version that you have configured the client to use (see "Updating the Client to Use a Later API Version," page 128). For example, if you have configured the client to use version 1.17, the namespace must be `urn:schemas-cybersource-com:transaction-data-1.17`.

Note The XML document that you receive in the reply always uses a prefix of `c:` (for example, `xmlns:c="urn:schemas-cybersource-com:transaction-data-1.17"`). Make sure you use an XML parser that supports namespaces.

Adding the Merchant ID

Optionally, you can add the CyberSource merchant ID to the request:

```
<?xml version="1.0" encoding="utf-8"?>
<requestMessage xmlns="urn:schemas-cybersource-com:transaction-data-
1.17">
  <merchantID>infodev</merchantID>
</requestMessage>
```

Adding Services to the Request

You next indicate the service that you want to use by creating an element for that service in the request, then setting the element's `run` attribute to `true`. For example, to request a credit card authorization:

```
<?xml version="1.0" encoding="utf-8"?>
<requestMessage xmlns="urn:schemas-cybersource-com:transaction-data-
1.15">
  <merchantID>infodev</merchantID>
  <ccAuthService run="true"/>
</requestMessage>
```

Requesting a Sale

You can request multiple services by creating additional elements. For example, if you fulfill the order immediately, you can request a credit card authorization and capture together (referred to as a “sale”):

```
<?xml version="1.0" encoding="utf-8"?>
<requestMessage xmlns="urn:schemas-cybersource-com:transaction-data-
1.17">
  <merchantID>infodev</merchantID>
  <ccAuthService run="true"/>
  <ccCaptureService run="true"/>
</requestMessage>
```

Adding Service-Specific Fields to the Request

You next add the fields that are used by the services you are requesting. Most fields are child elements of container elements; for example, a `<card>` element contains the customer's credit card information.

```
<?xml version="1.0" encoding="utf-8"?>
<requestMessage xmlns="urn:schemas-cybersource-com:transaction-data-
1.15">
  <merchantID>infodev</merchantID>
  <billTo>
    <firstName>Jane</firstName>
    <lastName>Smith</lastName>
  </billTo>
  <item id="0">
  </card>
    <unitPrice>29.95</unitPrice>
  </item>
  <card>
    <accountNumber>4111111111111111</accountNumber>
  <ccAuthService run="true"/>
</requestMessage>
```

The previous example shows only a partial list of the fields you must send. Refer to "Related Documents," page 21 for information about the guides that list all of the fields for the services that you are requesting.

Sending the Request

Once you have created an XML request document, you can use a .NET application to send the request to CyberSource. The example that follows is written in C#.

Note

The code in this section's examples is incomplete. For complete sample programs, see the source code in the client's `Xml\SingleServiceSample\` and `Xml\MultiServiceSample\` directories.

Creating a New Visual Studio .NET Project

To get started, create a new project in Visual Studio .NET, and add a reference to the client library, `XmlClient.dll`, which is located in the client's installation directory.

Importing the Client Classes

In the code for your application, add the following import statements:

```
using CyberSource.Xml;
using System;
using System.Net;
using System.Xml;
```

Sending the Request

You next read the XML request document, send the request to CyberSource, store the reply in a new `XmlDocument` object, and catch several exceptions that you might receive:

```
try {
  XmlDocument request = new XmlDocument();
  request.Load( "MyXmlDocument.xml" );

  XmlDocument reply = Client.RunTransaction( request );
  SaveOrderState();
  // "Using the Decision and Reason Code," page 135 describes the
  // ProcessReply method.
  ProcessReply( reply );
} catch (SignException se) {
  SaveOrderState();
  Console.WriteLine( se.ToString() );
} catch (FaultException fe) {
  SaveOrderState();
  /*
   * Some types of FaultException indicate that the transaction may
   * have been completed by CyberSource. The sample code shows how to
   * identify these exceptions. If you receive such an exception, and
   * your request included a payment service, you should use the
   * CyberSource transaction search screens to determine whether the
   * transaction was processed.
   */
  Console.WriteLine( fe.ToString() );
} catch (WebException we) {
  SaveOrderState();
  /*
   * Some types of WebException indicate that the transaction may
   * have been completed by CyberSource. The sample code shows how to
   * identify these exceptions. If you receive such an exception, and
   * your request included a payment service, you should use the
   * CyberSource transaction search screens to determine whether the
   * transaction was processed.
   */
  Console.WriteLine( we.ToString() );
}
```

```
private static void SaveOrderState() {
  /*
   * This is where you store the order state in your system for
   * post-transaction analysis. Be sure to store the customer
   * information, the values of the reply fields, and the details
   * of any exceptions that occurred.
   */
}
```

In the preceding example, when an exception occurs, the exception is printed to the console. Your web store should also display a message to the customer indicating that you were unable to process the order. The sample code for the XML client shows you how to provide feedback to the customer.

Also, if the transaction fails, and the request did not include any payment services, you may be able to resend the transaction. The sample code for the XML client shows you how to do this.

Interpreting the Reply

After the CyberSource server processes your request, it sends a reply message that contains information about the services you requested. You receive different fields depending on the services you request and the outcome of each service.

To use the reply information, you must integrate it into your system and any other system that uses that data. For example, you can store the reply information in a database and send it to other back office applications.

You must write an error handler to process the reply information that you receive from CyberSource. Do not show the reply information directly to customers. Instead, present an appropriate response that tells customers the result.

Important Because CyberSource may add reply fields and reason codes at any time, you should parse the reply data according to the names of the fields instead of their order in the reply.

The most important reply fields to evaluate are the following:

- **decision**: A one-word description of the results of your request. The decision is one of the following:
  - `ACCEPT` if the request succeeded
  - `REJECT` if one or more of the services in the request was declined
  - `REVIEW` if you use CyberSource Decision Manager and it flags the order for review. See "For CyberSource Advanced Merchants: Handling Decision Manager Reviews," page 149 for more information.
  - `ERROR` if there was a system error. See "Retrying When System Errors Occur," page 151 for important information about handling retries in the case of system errors.
- **reasonCode**: A numeric code that provides more specific information about the results of your request.

You also receive a reason code for each service in your request. You can use these reason codes to determine whether a specific service succeeded or failed. If a service fails, other services in your request may not run. For example, if you request a credit card authorization and capture, and the authorization fails, the capture does not run. The reason codes for each service are described in the *Credit Card Services User Guide* for CyberSource Essentials merchants or in the service developer guide for CyberSource Advanced merchants.

Important CyberSource reserves the right to add new reason codes at any time. If your error handler receives a reason code that it does not recognize, it should use the decision to interpret the reply.

Using the Decision and Reason Code

The following example shows how you can use the decision and the reason code to display an appropriate message to your customer.

```
private static bool ProcessReply( XmlDocument reply ) {
  // The following code allows you to use XPath with the CyberSource
  // schema, which uses a non-empty default namespace.
  XmlNamespaceManager nsmgr
    = new XmlNamespaceManager( reply.NameTable );
  nsmgr.AddNamespace( "cybs", Client.CYBS_NAMESPACE );

  XmlNode replyMessage
    = reply.SelectSingleNode( "cybs:replyMessage", nsmgr);

  string decision = replyMessage.SelectSingleNode(
    "cybs:decision/text()", nsmgr ).Value;
  string template = GetTemplate( decision.ToUpper() );
  string content  = GetContent( replyMessage, nsmgr );

  // This example writes the message to the console. Choose an
  // appropriate display method for your own application.
  Console.WriteLine( template, content );
}

private static string GetTemplate( string decision ) {
  // Retrieves the text that corresponds to the decision.
  if ("ACCEPT".Equals( decision )) {
    return( "The order succeeded.{0}" );
  }

}
  if ("REJECT".Equals( decision )) {
    return( "Your order was not approved.{0}" );
  }

  // ERROR, or an unknown decision
  return( "Your order could not be completed at this time.{0}" +
          "\nPlease try again later." );
private static string GetContent(
  XmlNode replyMessage, XmlNamespaceManager nsmgr ) {
  /*
   * Uses the reason code to retrieve more details to add to the
   * template.
   *
   * The messages returned in this example are meant to demonstrate
   * how to retrieve the reply fields. Your application should
   * display user-friendly messages.
   */
  string textVal = replyMessage.SelectSingleNode(
    "cybs:reasonCode/text()", nsmgr ).Value;
  int reasonCode = int.Parse( textVal );
  switch (reasonCode) {

    // Success
    case 100:
      return( "\nRequest ID: " +
        replyMessage.SelectSingleNode(
          "cybs:requestID/text()", nsmgr ).Value );

    // Missing field or fields
    case 101:
      return( "\nThe following required fields are missing: " +
              EnumerateValues( replyMessage.SelectNodes(
                "cybs:missingField/text()", nsmgr ) ) );

    // Invalid field or fields
    case 102:
      return( "\nThe following fields are invalid: " +
              EnumerateValues( replyMessage.SelectNodes(
                "cybs:invalidField/text()", nsmgr ) ) );

    // Insufficient funds
    case 204:
      return( "\nInsufficient funds in the account. Please use a " +
              "different card or select another form of payment." );

    // Add additional reason codes here that you must handle
    // more specifically.

    default:
      // For all other reason codes (for example, unrecognized reason
      // codes, or codes that do not require special handling),
      // return an empty string.
      return( String.Empty );
  }
}

private static string EnumerateValues( XmlNodeList nodes ) {
  System.Text.StringBuilder sb = new System.Text.StringBuilder();
  foreach (XmlNode node in nodes) {
    sb.Append( val + "\n" );
  }
  return( sb.ToString() );
}
```

For CyberSource Advanced Merchants: Handling Decision Manager Reviews

The information in this section applies only to CyberSource Advanced merchants.

If you use CyberSource Decision Manager, you may also receive the `REVIEW` value in the **decision** field. `REVIEW` means that Decision Manager has marked the order for review based on how you configured the Decision Manager rules.

If you will be using Decision Manager, you have to determine how to handle the new `REVIEW` value. Ideally, you will update your order management system to recognize the `REVIEW` response and handle it according to your business rules. If you cannot update your system to handle the `REVIEW` response, CyberSource recommends that you choose one of these options:

- If you authorize and capture the credit card payment at the same time, treat the `REVIEW` response like a `REJECT` response. Rejecting any orders that are marked for review may be appropriate if your product is a software download or access to a Web site. If supported by your processor, you may also want to reverse the authorization.

- If you approve the order after reviewing it, convert the order status to `ACCEPT` in your order management system. You can request the credit card capture without requesting a new authorization.

- If you approve the order after reviewing it but cannot convert the order status to `ACCEPT` in your system, request a new authorization for the order. When processing this new authorization, you must disable Decision Manager. Otherwise the order will be marked for review again. For details about the API field that disables Decision Manager, see the *Decision Manager Developer Guide Using the Simple Order API* (PDF | HTML) or the *Decision Manager Developer Guide Using the SCMP Order API* (PDF | HTML).

Alternately, you can specify a custom business rule in Decision Manager so that authorizations originating from a particular internal IP address at your company are automatically accepted.

If supported by your processor, you may want to reverse the original authorization.

Requesting Multiple Services

When you request multiple services in one request, CyberSource processes the services in a specific order. If a service fails, CyberSource does not process the subsequent services in the request.

For example, in the case of a sale (a credit card authorization and a capture requested together), if the authorization service fails, CyberSource will not process the capture service. The reply you receive only includes reply fields for the authorization.

Many CyberSource services include “ignore” fields that tell CyberSource to ignore the result from the first service when deciding whether to run the subsequent services. In the case of the sale, even though the issuing bank gives you an authorization code, CyberSource might decline the authorization based on the AVS or card verification results. Depending on your business needs, you might choose to capture these types of declined authorizations anyway. You can set the **businessRules_ignoreAVSResult** field to `"true"` in your combined authorization and capture request:

```
<businessRules>
    <ignoreAVSResult>true</ignoreAVSResult>
</businessRules>
```

This tells CyberSource to continue processing the capture even if the AVS result causes CyberSource to decline the authorization. In this case you would then get reply fields for both the authorization and the capture in your reply.

Note You are charged only for the services that CyberSource performs.

Retrying When System Errors Occur

You must design your transaction management system to include a way to correctly handle CyberSource system errors. Depending on which payment processor is handling the transaction, the error may indicate a valid CyberSource system error, or it may indicate a processor rejection because of some type of invalid data. In either case, CyberSource recommends that you do not design your system to retry sending a transaction many times in the case of a system error.

Instead, CyberSource recommends that you retry sending the request only two or three times with successively longer periods of time between each retry. For example, after the first system error response, wait 30 seconds and then retry sending the request. If you receive the same error a second time, wait one minute before you send the request again. Depending on the situation, you may decide you can retry sending the request after a longer time period. Determine what is most appropriate for your business situation.

If after several retry attempts you are still receiving a system error, it is possible that the error is actually being caused by a processor rejection and not a CyberSource system error. In that case, we suggest that you either:

- Search for the transaction in the Business Center, look at the description of the error on the Transaction Detail page, and call your processor to determine if and why they are rejecting the transaction.

- Contact CyberSource Customer Support to confirm whether your error is truly caused by a CyberSource system issue.

If TSYS Acquiring Solutions is your processor, you may want to follow the first suggestion as there are several common TSYS Acquiring Solutions processor responses that are returned to you as system errors and that only TSYS Acquiring Solutions can address.

Creating an Application Settings File

After you finish writing code for your integration, you must create an application settings file. This file must contain, at a minimum, the following information:

- The directory that contains your security key
- The location of the CyberSource server

See Table 33, "Fields in the Settings File," on page 123 for a complete list of settings.

You can use the settings files that come with the sample applications as a starting point for your own settings file. See "Configuring the Test Applications," page 123 for more information.

Using SOAP

This section explains how to request CyberSource services by using the SOAP (Simple Object Access Protocol).

Requesting CyberSource Services

To request CyberSource services, write code that:

- Collects information for the CyberSource services that you will use
- Assembles the order information into requests
- Sends the requests to the CyberSource server

> ⚠ **Important** The CyberSource servers do not support persistent HTTP connections.

- Processes the reply information

The instructions in this section explain how to write C# programs that request CyberSource services. For a list of API fields to use in your requests, see "Related Documents," page 21.

Sample Code

The code in this section's examples is incomplete. For complete sample programs, see the source code in the client's `Soap\SingleServiceSample` and `Soap\MultiServiceSample` directories.

Creating and Sending the Request

To use any CyberSource service, you must create and send a request that includes the required information for that service.

The following example shows basic code for requesting CyberSource services. In this example, Jane Smith is buying an item for $29.95.

Creating a New Visual Studio .NET Project

To get started, create a new project in Visual Studio .NET, and add a reference to the client library, `SoapClient.dll`, which is located in the client's installation directory. You must also add a reference to the library `System.Web.Services.dll`, which is part of the .NET Framework.

Importing the Client Classes

In the code for your application, add the following import statements:

```
using CyberSource.Soap;
using CyberSource.Soap.CyberSourceWS;
using System;
using System.Configuration;
using System.Net;
using System.Web.Services.Protocols;
```

Creating an Empty Request

You next create a RequestMessage object that holds the request fields:

```
RequestMessage request = new RequestMessage();
```

Adding the Merchant ID

You next optionally add your CyberSource merchant ID to the request:

```
request.merchantID = "infodev";
```

This value overrides any value you set with the merchantID configuration setting (see Table 33, "Fields in the Settings File," on page 123).

Adding Services to the Request

You next indicate the service that you want to use by creating an object for that service in the request, then setting the object's `run` property to `true`. For example, to request a credit card authorization:

```
request.ccAuthService = new CCAuthService();
request.ccAuthService.run = "true";
```

Requesting a Sale

You can request multiple services by creating additional objects. For example, if you fulfill the order immediately, you can request a credit card authorization and capture together (referred to as a “sale”):

```
request.ccAuthService = new CCAuthService();
request.ccAuthService.run = "true";
request.ccCaptureService = new CCCaptureService();
request.ccCaptureService.run = "true";
```

Adding Service-Specific Fields to the Request

You next add the fields that are used by the services you are requesting. Most fields are properties of additional objects; for example, a `Card` object contains the customer’s credit card information.

```
BillTo billTo = new BillTo();
billTo.firstName = "Jane";
billTo.lastName = "Smith";
request.billTo = billTo;

Card card = new Card();
card.accountNumber = "4111111111111111";
request.card = card;

// there is one item in this sample
request.item = new Item[1];
Item item = new Item();
item.id = "0";
item.unitPrice = "29.95";
request.item[0] = item;
```

The preceding example shows only a partial list of the fields you must send. Refer to "Related Documents," page 21 for information about the guides that list all of the fields for the services that you are requesting.

Sending the Request

You next send the request to CyberSource, store the reply in a new `ReplyMessage` object, and handle several exceptions that you might receive.

```
try {
  ReplyMessage reply = Client.RunTransaction( request );
  SaveOrderState();

  // "Using the Decision and Reason Code," page 135 describes the
  // ProcessReply method.
  ProcessReply( reply );
} catch (SignException se) {
  SaveOrderState();
  Console.WriteLine( se.ToString() );
} catch (SoapHeaderException she) {
  SaveOrderState();
  Console.WriteLine( she.ToString() );
} catch (SoapBodyException sbe) {
  SaveOrderState();
  /*
   * Some types of SoapBodyException indicate that the transaction may
   * have been completed by CyberSource. The sample code shows how to
   * identify these exceptions. If you receive such an exception, and
   * your request included a payment service, you should use the
   * CyberSource transaction search screens to determine whether the
   * transaction was processed.
   */
  Console.WriteLine( sbe.ToString() );
} catch (WebException we) {
  SaveOrderState();
  /*
   * Some types of WebException indicate that the transaction may
   * have been completed by CyberSource. The sample code shows how to
   * identify these exceptions. If you receive such an exception, and
   * your request included a payment service, you should use the
   * CyberSource transaction search screens to determine whether the
   * transaction was processed.
   */
  Console.WriteLine( we.ToString() );
}
private static void SaveOrderState() {
  /*
   * This is where you store the order state in your system for
   * post-transaction analysis. Be sure to store the customer
   * information, the values of the reply fields, and the details
   * of any exceptions that occurred.
   */
}
```

In the preceding example, when an exception occurs, the exception is printed to the console. Your web store should also display a message to the customer indicating that you were unable to process the order. The sample code for the SOAP client shows you how to provide feedback to the customer.

Also, if the transaction fails, and the request did not include any payment services, you may be able to resend the transaction. The sample code for the SOAP client shows you how to do this.

Interpreting the Reply

After the CyberSource server processes your request, it sends a reply message that contains information about the services you requested. You receive different fields depending on the services you request and the outcome of each service.

To use the reply information, you must integrate it into your system and any other system that uses that data. For example, you can store the reply information in a database and send it to other back office applications.

You must write an error handler to process the reply information that you receive from CyberSource. Do not show the reply information directly to customers. Instead, present an appropriate response that tells customers the result.

Important Because CyberSource may add reply fields and reason codes at any time, you should parse the reply data according to the names of the fields instead of their order in the reply.

The most important reply fields to evaluate are the following:

- **decision**: A one-word description of the results of your request. The decision is one of the following:
  - `ACCEPT` if the request succeeded
  - `REJECT` if one or more of the services in the request was declined
  - `REVIEW` if you use CyberSource Decision Manager and it flags the order for review. See "For CyberSource Advanced Merchants: Handling Decision Manager Reviews," page 159 for more information.
  - `ERROR` if there was a system error. See "Retrying When System Errors Occur," page 161 for important information about handling retries in the case of system errors.
- **reasonCode**: A numeric code that provides more specific information about the results of your request.

You also receive a reason code for each service in your request. You can use these reason codes to determine whether a specific service succeeded or failed. If a service fails, other services in your request may not run. For example, if you request a credit card

authorization and capture, and the authorization fails, the capture does not run. The reason codes for each service are described in the *Credit Card Services User Guide* for CyberSource Essentials merchants or in the service developer guide for CyberSource Advanced merchants.

Important CyberSource reserves the right to add new reason codes at any time. If your error handler receives a reason code that it does not recognize, it should use the decision to interpret the reply.

Using the Decision and Reason Code

The following example shows how you can use the decision and the reason code to display an appropriate message to your customer.

```
private static bool ProcessReply( ReplyMessage reply ) {
  string template = GetTemplate( reply.decision.ToUpper() );
  string content  = GetContent( reply );

  // This example writes the message to the console. Choose an
  // appropriate display method for your own application.
  Console.WriteLine( template, content );
}
private static string GetTemplate( string decision ) {

  // Retrieves the text that corresponds to the decision.
  if ("ACCEPT".Equals( decision )) {
    return( "The order succeeded.{0}" );
  }
  if ("REJECT".Equals( decision )) {
    return( "Your order was not approved.{0}" );

  }

  // ERROR, or an unknown decision
  return( "Your order could not be completed at this time.{0}" +
          "\nPlease try again later." );
}
private static string GetContent( ReplyMessage reply ) {
  /*
   * Uses the reason code to retrieve more details to add to the
   * template.
   *
   * The messages returned in this example are meant to demonstrate
   * how to retrieve the reply fields. Your application should
   * display user-friendly messages.
   */

  int reasonCode = int.Parse( reply.reasonCode );
  switch (reasonCode) {

    // Success
    case 100:
      return( "\nRequest ID: " + reply.requestID );

    // Missing field or fields
    case 101:
      return( "\nThe following required fields are missing: " +
              EnumerateValues( reply.missingField ) );

    // Invalid field or fields
    case 102:
      return( "\nThe following fields are invalid: " +
              EnumerateValues( reply.invalidField ) );

    // Insufficient funds
    case 204:
      return( "\nInsufficient funds in the account. Please use a " +
              "different card or select another form of payment." );

    // Add additional reason codes here that you must handle
    // more specifically.
    default:

      // For all other reason codes (for example, unrecognized reason
      // codes, or codes that do not require special handling),
      // return an empty string.
      return( String.Empty );
  }
}

private static string EnumerateValues( string[] array ) {
  System.Text.StringBuilder sb = new System.Text.StringBuilder();
  foreach (string val in array) {
    sb.Append( val + "\n" );
  }
  return( sb.ToString() );
}
```

For CyberSource Advanced Merchants: Handling Decision Manager Reviews

The information in this section applies only to CyberSource Advanced merchants.

If you use CyberSource Decision Manager, you may also receive the `REVIEW` value in the **decision** field. `REVIEW` means that Decision Manager has marked the order for review based on how you configured the Decision Manager rules.

If you will be using Decision Manager, you have to determine how to handle the new `REVIEW` value. Ideally, you will update your order management system to recognize the `REVIEW` response and handle it according to your business rules. If you cannot update your system to handle the `REVIEW` response, CyberSource recommends that you choose one of these options:

- If you authorize and capture the credit card payment at the same time, treat the `REVIEW` response like a `REJECT` response. Rejecting any orders that are marked for review may be appropriate if your product is a software download or access to a Web site. If supported by your processor, you may also want to reverse the authorization.

- If you approve the order after reviewing it, convert the order status to `ACCEPT` in your order management system. You can request the credit card capture without requesting a new authorization.

- If you approve the order after reviewing it but cannot convert the order status to `ACCEPT` in your system, request a new authorization for the order. When processing this new authorization, you must disable Decision Manager. Otherwise the order will be marked for review again. For details about the API field that disables Decision Manager, see the *Decision Manager Developer Guide Using the Simple Order API* (PDF | HTML) or the *Decision Manager Developer Guide Using the SCMP Order API* (PDF | HTML).

Alternately, you can specify a custom business rule in Decision Manager so that authorizations originating from a particular internal IP address at your company are automatically accepted.

If supported by your processor, you may want to reverse the original authorization.

Requesting Multiple Services

When you request multiple services in one request, CyberSource processes the services in a specific order. If a service fails, CyberSource does not process the subsequent services in the request.

For example, in the case of a sale (a credit card authorization and a capture requested together), if the authorization service fails, CyberSource will not process the capture service. The reply you receive only includes reply fields for the authorization.

Many CyberSource services include “ignore” fields that tell CyberSource to ignore the result from the first service when deciding whether to run the subsequent services. In the case of the sale, even though the issuing bank gives you an authorization code, CyberSource might decline the authorization based on the AVS or card verification results. Depending on your business needs, you might choose to capture these types of declined authorizations anyway. You can set the **businessRules_ignoreAVSResult** field to `"true"` in your combined authorization and capture request:

```
BusinessRules businessRules = new BusinessRules();

businessRules.ignoreAVSResult = "true";

request.businessRules = businessRules;
```

This tells CyberSource to continue processing the capture even if the AVS result causes CyberSource to decline the authorization. In this case you would then get reply fields for both the authorization and the capture in your reply.

Note

You are charged only for the services that CyberSource performs.

Retrying When System Errors Occur

You must design your transaction management system to include a way to correctly handle CyberSource system errors. Depending on which payment processor is handling the transaction, the error may indicate a valid CyberSource system error, or it may indicate a processor rejection because of some type of invalid data. In either case, CyberSource recommends that you do not design your system to retry sending a transaction many times in the case of a system error.

Instead, CyberSource recommends that you retry sending the request only two or three times with successively longer periods of time between each retry. For example, after the first system error response, wait 30 seconds and then retry sending the request. If you receive the same error a second time, wait one minute before you send the request again. Depending on the situation, you may decide you can retry sending the request after a longer time period. Determine what is most appropriate for your business situation.

If after several retry attempts you are still receiving a system error, it is possible that the error is actually being caused by a processor rejection and not a CyberSource system error. In that case, we suggest that you either:

- Search for the transaction in the Business Center, look at the description of the error on the Transaction Detail page, and call your processor to determine if and why they are rejecting the transaction.

- Contact CyberSource Customer Support to confirm whether your error is truly caused by a CyberSource system issue.

If TSYS Acquiring Solutions is your processor, you may want to follow the first suggestion as there are several common TSYS Acquiring Solutions processor responses that are returned to you as system errors and that only TSYS Acquiring Solutions can address.

Creating an Application Settings File

After you finish writing code for your integration, you must create an application settings file. This file must contain, at a minimum, the following information:

- The directory that contains your security key
- The location of the CyberSource server

See Table 33, "Fields in the Settings File," on page 123 for a complete list of settings.

You can use the settings files that come with the sample applications as a starting point for your own settings file. See "Configuring the Test Applications," page 123 for more information.

Setting the Connection Limit

This appendix explains how to increase the number of simultaneous connections between the client and CyberSource.

By default, you can create only two simultaneous connections to an HTTP server. By increasing the number of connections, you can avoid a backlog of requests during times of very high transaction volume. Microsoft recommends for the connection limit a value that is 12 times the number of CPUs. For example, if you have two CPUs, you can set the connection limit to 24. To determine the optimum setting for your application, make sure to run performance tests.

Examples

You can increase the number of connections in many ways, for example by using an application- or server-specific configuration file where you can change the setting for a single or for all hosts. The examples below describe briefly some of the methods that you can use to increase connection limits.

cybs.connectionLimit

When set to a value other than `-1`, the `cybs.connectionLimit` setting in the client increases the limit for the host where you are sending the request by executing these statements on your behalf:

```
ServicePoint sp = ServicePointManager.FindServicePoint(uri);
sp.ConnectionLimit = config.ConnectionLimit;
```

<connectionManagement>

You can set the connection limit by using .NET's `<connectionManagement>` tag. In this example, the connection limit for CyberSource's test and production hosts is 12 while the limit for all other hosts is 2:

```
<system.net>
    <connectionManagement>
        <add address = "https://ics2wstesta.ic3.com" maxconnection = "12"
        />
        <add address = "https://ics2wsa.ic3.com" maxconnection = "12" />
        <add address = "*" maxconnection = "2" />
    </connectionManagement>
</system.net>
```

DefaultConnectionLimit

You can set the connection limit for all hosts to which your application is connected before a connection is made by using the following line in your start-up code:

```
ServicePointManager.DefaultConnectionLimit = your_value_here;
```

References

For more information on these and other methods to increase the connection limits, see the following Microsoft documentation:

Managing Connections in the .Net Framework Developer's Guide (`http://msdn2.microsoft.com/en-us/library/7af54za5.aspx`).

CHAPTER 5

.NET 2.0 Client

Important

- The .NET 2.0 client for the Simple Order API is supported on 32-bit operating systems only.
- If you are building an application to sell to others, see Appendix A, "Using the Client Application Fields," on page 390. This appendix has a list of API fields you can use in your request that describe the application, its version, and its user. If you use these fields in your request, you can view their values in the Transaction Search Details window of the Business Center.

Choosing an API and Client

API Variation

With this client package, you can use any of the three variations of the Simple Order API:

- Name-value pairs, which are simpler to use than XML
- XML, which requires you to create and parse XML documents
- SOAP (Simple Object Access Protocol) 1.1, which provides an object-oriented interface

The test that you run immediately after installing the client uses name-value pairs.

Client Versions

CyberSource updates the Simple Order API on a regular basis to introduce new API fields and functionality. To identify the latest version of the API, go to https://ics2wsa.ic3.com/commerce/1.x/transactionProcessor.

This represents the version of the server-side code for the CyberSource services.

Note Starting with version 5.0.0, the version of the Simple Order API client for .NET no longer matches the version of the server API that it uses. See the README file for the correct server API version.

If a new version of the API has been released, but CyberSource has not yet updated the .NET client to use this new version, you can manually update the client to use a different version. See "Updating the Client to Use a Later API Version," page 176.

Basic C# Program Example

The following example shows the primary code required to send a SOAP request for credit card authorization and process the reply. See "Using SOAP," page 200 for more information.

```
using CyberSource.Soap;
using CyberSource.Soap.CyberSourceWS;
using System;
using System.Configuration;
using System.Net;
using System.Web.Services.Protocols;
namespace Sample {
  class Sample {
    static void Main(string[] args) {
      RequestMessage request = new RequestMessage();
      request.merchantID = "infodev";

      // we want to do Credit Card Authorization in this sample
      request.ccAuthService = new CCAuthService();
      request.ccAuthService.run = "true";
      // add required fields
      request.merchantReferenceCode = "148705832705344";
      BillTo billTo = new BillTo();
      billTo.firstName = "Jane";
      billTo.lastName = "Smith";
      billTo.street1 = "1295 Charleston Road";
      billTo.city = "Mountain View";
      billTo.state = "CA";
      billTo.postalCode = "94043";
      billTo.country = "US";
      billTo.email = "jsmith@example.com";
      request.billTo = billTo;
      Card card = new Card();
      card.accountNumber = "4111111111111111";
      card.expirationMonth = "12";
      card.expirationYear = "2010";
      request.card = card;
      PurchaseTotals purchaseTotals = new PurchaseTotals();
      purchaseTotals.currency = "USD";
      request.purchaseTotals = purchaseTotals;

      // there is one item in this sample
      request.item = new Item[1];
      Item item = new Item();
      item.id = "0";
      item.unitPrice = "29.95";
      request.item[0] = item;
      // See "Interpreting the Reply," page 204 for details about

      // processing the reply for a SOAP transaction.
      try {
        ReplyMessage reply = Client.RunTransaction( request );
      } catch (SignException se) {
        Console.WriteLine( se.ToString() );
      } catch (SoapHeaderException she) {
        Console.WriteLine( she.ToString() );
      } catch (SoapBodyException sbe) {
        Console.WriteLine( sbe.ToString() );
      } catch (WebException we) {
        Console.WriteLine( we.ToString() );
      }
    }
  }
}
```

Installing and Testing the Client

Minimum System Requirements

- Microsoft Windows 2000 or later
- .NET Framework 2.0 or later
- Microsoft Web Services Enhancements (WSE) 3.0 or later
- Microsoft Visual Studio 2005

Important

Failure to configure your client API host to a unique, public IP address will cause inconsistent transaction results.

The client API request ID algorithm uses a combination of IP address and system time, along with other values. In some architectures this combination might not yield unique identifiers.

Transaction Security Keys

The first thing you must do is create your security key. The client uses the security key to add a digital signature to every request that you send. This signature helps ensure that no one else can use your CyberSource account to process orders. You specify the location of your key when you configure the client.

Important

You must generate two transaction security keys—one for the CyberSource production environment and one for the test environment. For information about generating and using security keys, see *Creating and Using Security Keys* (PDF | HTML).

Warning

You must protect your security key to ensure that your CyberSource account is not compromised.

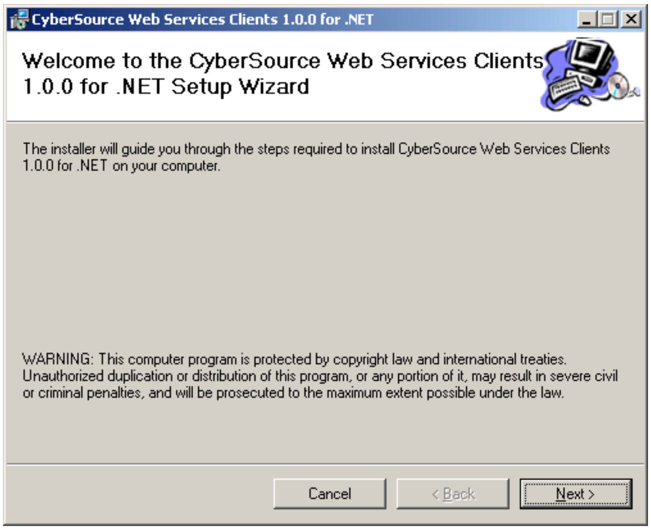
Installing for the First Time

To install the client for the first time:

Step 1 Go to the client downloads page on the Support Center and download the latest version of the client.

Step 2 Run the downloaded file.

The Simple Order API for .NET Setup Wizard appears.



CyberSource Web Services Clients 1.0.0 for .NET

Welcome to the CyberSource Web Services Clients 1.0.0 for .NET Setup Wizard

The installer will guide you through the steps required to install CyberSource Web Services Clients 1.0.0 for .NET on your computer.

WARNING: This computer program is protected by copyright law and international treaties. Unauthorized duplication or distribution of this program, or any portion of it, may result in severe civil or criminal penalties, and will be prosecuted to the maximum extent possible under the law.

Step 3 Follow the instructions in the wizard to install the client.

If you use the default installation directory, the client is installed in `c:\simapi-net-2.0-<version>`.

Step 4 Test the client. See "Using the Test Applications," page 170.

The client is installed and tested. You are ready to create your own code for requesting CyberSource services. Finish reading this section, and move on to one of these sections:

- "Using Name-Value Pairs," page 178 if you plan to use name-value pairs
- "Using XML," page 188 if you plan to use XML
- "Using SOAP," page 200 if you plan to use SOAP

Upgrading from a Previous Version

To upgrade your client from a previous version:

Step 1 Follow the previous installation instructions in "Installing for the First Time," page 168.

Step 2 In Visual Studio, remove the reference to the previous client.

Step 3 Add a reference to the new client (`CyberSource.Clients.dll`).

You have successfully upgraded your client to the new version.

Migrating from .NET Framework 1.x

If you are switching from .NET Framework 1.x and are using a CyberSource Web Services client or CyberSource Simple Order API client, you must use the following migration procedure:

To migrate from a .NET Framework 1.x client to a 2.0 client:

Step 1 Replace the old DLLs with the ones from this package.

Step 2 In your project, remove references to the previous CyberSource DLLs.

Step 3 Add a reference to `CyberSource.Clients.dll`.

Step 4 In your request code, make the following changes:

a Replace the referenced CyberSource namespaces with this one:

```
CyberSource.Clients
```

b If you use the SOAP client, add the following namespace:

```
CyberSource.Clients.SoapWebReference
```

Example **In C#, with the SOAP client, you now have:**

```
using CyberSource.Clients.
using CyberSource.Clients.SoapWebReference; /* for SOAP client
only */
```

c Replace `Client.RunTransaction` with the call appropriate for your API:

```
NVPClient.RunTransaction
XmlClient.RunTransaction
SoapClient.RunTransaction
```

Step 5 To prevent any conflicts with any of your other settings, add the prefix `cybs.` to your previous configuration setting keys.

For example, see Table 34, "Fields in the Settings File," on page 171. Note that the previous configuration setting keys are still supported.

Step 6 Replace the key `cybersourceURL` with `cybs.sendToProduction` and set it to `true` or `false` (or `0` or `1`) as appropriate.

For a list of new and modified configuration settings, see Table 34, "Fields in the Settings File," on page 171. Use the sample applications provided as reference during the migration.

Using the Test Applications

Each type of client variation—name-value pair, XML, and SOAP—includes a pre-compiled test application. You can use these test applications to ensure that the client was installed correctly. The applications request both credit card authorization and capture.

The test applications and their source code are installed in the `samples` directory. The `bin` subdirectory contains the pre-compiled binaries. The `src` subdirectory contains the source code and Visual Studio project files.

Configuring the Test Applications

Before you run a test application, you must edit its application settings file. The following table describes all the configuration fields that you can use in this file.

Note Configuration settings supported by the latest 1.x.x version are still supported. However, CyberSource recommends that you use the following new settings for this and future versions.

Table 34 Fields in the Settings File

Field Name	Description	Required/ Optional
cybs.connectionLimit	Maximum number of allowed concurrent connections between the client and CyberSource's server. For more information on this field and alternate ways to set the connection limits, see "Setting the Connection Limit," page 210.	Optional
cybs.keysDirectory	Directory that contains the pkcs12 security key file. For example: `c:\keys\`	Required
cybs.merchantID	Your CyberSource merchant ID. You can override this value by providing the merchantID field in the request itself. The merchant ID is case sensitive.	Optional
cybs. sendToProduction	Flag that indicates whether the transactions for this merchant should be sent to the production server. Use one of these values: ■ `false`: Do not send to the production server; send to the test server (default setting). ■ `true`: Send to the production server. **Note** Make sure that if your merchant ID is configured to use the test mode, you send requests to the test server.	Required
cybs.keyFilename	Name of the security key file name for the merchant in the format `<`***`security_key_filename`***`>.p12`.	Optional
cybs.serverURL	Alternate server URL to use. For more information, see "Configuring Your Settings for Multiple Merchants," page 173. Give the complete URL because it will be used exactly as you specify.	Optional

Table 34 Fields in the Settings File (Continued)

Field Name	Description	Required/ Optional
cybs.enableLog	Flag directing the client to log transactions and errors. Use one of these values: ▪ `false`: Do not enable logging (default setting). ▪ `true`: Enable logging. **Important** Logging can cause very large log files to accumulate. Therefore, CyberSource recommends that you use logging only when troubleshooting problems. To comply with all Payment Card Industry (PCI) and Payment Application (PA) Data Security Standards regarding the storage of credit card and card verification number data, the logs that are generated contain only masked credit card and card verification number data (CVV, CVC2, CVV2, CID, CVN). Follow these guidelines: ▪ Use debugging temporarily for diagnostic purposes only. ▪ If possible, use debugging only with test credit card numbers. ▪ Never store clear text card verification numbers. ▪ Delete the log files as soon as you no longer need them. ▪ Never send email to CyberSource containing personal and account information, such as customers' names, addresses, card or check account numbers, and card verification numbers. For more information about PCI and PABP requirements, see www.visa.com/cisp.	Optional
cybs.logDirectory	Directory to which to write the log file. Note that the client will not create this directory for you; you must specify an existing directory. The client includes a `logs` directory that you can use. Include the path. For example: `c:\simapi-net-2.0.0\logs`.	Required if **cybs. enableLog** is true
cybs.logFilename	Name of the log file. The client uses `cybs.log` by default.	Optional
cybs.logMaximumSize	Maximum size in megabytes for the log file. The default value is `10`. When the log file reaches the specified size, it is archived into `cybs.log.`***<yyyymmdd***`T`***hhmmssxxx>*** and a new log file is started. The ***xxx*** indicates milliseconds.	Optional
cybs.timeout	Length of time-out in seconds. The default is 130.	Optional
cybs.proxyURL	URL of a proxy server. For example: `https://proxy.example.com:4909`	Optional
cybs.proxyUser	User name for the proxy server.	Optional
cybs.proxyPassword	Password for the proxy server.	Optional

To test applications:

Step 1 Decide which test application you want to run, such as `SoapSample.exe`.

Step 2 Using a text editor, open the settings file for the test application.

The settings file has the same name as the test application, with the extension `config` appended to the name. For example, `SoapSample.exe.config`.

Step 3 Find the `cybs.merchantID` field and change its value to your CyberSource merchant ID.

For example, if your merchant ID is `widgetsinc`, change the field to `<add key="cybs.merchantID" value="`**`widgetsinc`**`"/>`.

The merchant ID is case sensitive.

Step 4 Find the `cybs.keysDirectory` field and change its value to the directory that contains your security key.

For example, if your key is in `c:\keys\`, change the field to `<add key="cybs.keysDirectory" value="`**`c:\keys\`**`"/>`.

Step 5 Edit other fields as necessary.

See Table 34, "Fields in the Settings File," on page 171 for a complete list.

Step 6 Save and close the settings file.

Configuring Your Settings for Multiple Merchants

If you have multiple merchant IDs, or if you are a reseller handling multiple merchants, you can configure the settings to allow different configurations for different merchant IDs.

To specify the settings for a specific merchant, prefix all settings, except for `cybs.merchantID` and the `cybs.proxy*`, with `<merchantID>.` The `cybs.proxy*` wildcard refers to the `proxyURL`, `proxyUser`, `proxyPassword` settings.

Example **You have a new merchant with merchant ID of `NewMerchant`. To send only test transactions for this merchant, you can set all requests for `NewMerchant` to go to the test server:**

```
<add key="cybs.NewMerchant.sendToProduction" value="false"/>
<add key="cybs.sendToProduction" value="true"/>
```

With the second line of the example, the client will send all other requests to the production server.

Running the Test Applications

To run test applications:

Step 1 Open a Windows command-line shell.

Step 2 Change to the directory where the test application is located.

Step 3 Type the name of the test application, then press Enter.

The test application requests an CyberSource service, interprets the reply, and prints information about the result. If you receive a .NET exception, use the error message to debug the problem.

Deploying the Client to Another Computer

To deploy the client to another computer without running the installer provided by CyberSource, you must include all the files from the `lib` directory in your custom installer.

Then, you must register `CybsWSSecurity.dll` either in your installation script or on the command prompt.

To register CybsWSSecurity.dll:

Step 1 Open a command prompt.

Step 2 Go to the directory where you installed the client files.

Step 3 Type this line: **`regsvr32 CybsWSSecurity.dll`**

The client is now ready to be used on the computer.

Going Live

When you complete all of your system testing and are ready to accept real transactions from consumers, your deployment is ready to *go live*.

Note

After your deployment goes live, use real card numbers and other data to test every card type you support. Because these are real transactions in which you are buying from yourself, use small monetary amounts to do the tests. Process an authorization, then capture the authorization, and later refund the money. Use your bank statements to verify that money is deposited into and withdrawn from your merchant bank account as expected. If you have more than one CyberSource merchant ID, test each one separately.

CyberSource Essentials Merchants

If you use CyberSource Essentials services, you can use the Business Center site to go live. For a description of the process of going live, see the “Steps for Getting Started” section in *Getting Started with CyberSource Essentials*.

Important

You must also configure your client so that it sends transactions to the production server and not the test server. See the description of the configuration setting "cybs. sendToProduction," page 171.

CyberSource Advanced Merchants

If you use CyberSource Advanced services, see the “Steps for Getting Started” chapter in *Getting Started with CyberSource Advanced* for information about going live.

When your deployment goes live, your CyberSource account is updated so that you can send transactions to the CyberSource production server. If you have not already done so, you must provide your banking information to CyberSource so that your processor can deposit funds to your merchant bank account.

After CyberSource confirms that your account is live, make sure that you update your system so that it can send requests to the production server (`ics2wsa.ic3.com`) using your security key for the production environment. The test server (`ics2wstesta.ic3.com`) cannot be used for real transactions. For more information about sending transactions to the production server, see the description of the configuration setting "cybs. sendToProduction," page 171.

Updating the Client to Use a Later API Version

CyberSource periodically updates the Simple Order API. You can update your existing client to work with the new API version. For a list of the available API versions, go to https://ics2wsa.ic3.com/commerce/1.x/transactionProcessor/.

Alternately, if a new client is available that works with the later API version, you can download that new client.

Note The new client may have new functionality unrelated to the changes in the API. Read the release notes in the `CHANGES` file to determine if the new client contains new functionality that you want to use.

Name-Value Pair Client

To update a name-value pair client:

Step 1 Load `src\CyberSource.Clients.sln` in Visual Studio 2005.

Step 2 Enter the URL of the new WSDL file in the Web Reference URL of the NVPWebReference web reference. This can be done automatically by right-clicking NVPWebReference and choosing `Update Web Reference`. After a successful update, a new `Reference.cs` file is generated in the `Web References\NVPWebReference` directory.

Step 3 Load the newly-generated `Reference.cs` and look for the constructor for NVPTransactionProcessWse.

Step 4 Update the constructor so that it accepts the parameter `string url` and replace the hard-coded URL assigned to `this.URL` with this parameter. For example:

```
public NVPTransactionProcessorWse(string url) {

    this.Url = url;
```

Step 5 In the Project Properties, switch to the Signing tab and choose your own strong name key file or clear `Sign the assembly` if you do not want to sign the assembly. To generate your own key file, use the Strong Name tool (`sn.exe`) that is included with the .NET Framework SDK.

Step 6 Build the Release configuration. If this results in build errors related to NVPTransactionProcessorWse, do the following:

a Rerun the WSE 3.0 installer. You can do this by using Add/Remove Programs.

b In the installer, select `modify` and install the `Visual Studio Tools` option.

c Restart Visual Studio.

d Return to step 1.

Step 7 Save a copy of the original `CyberSource.Clients.dll` and then replace it with the newly built `CyberSource.Clients.dll`.

SOAP Client

To update a SOAP client:

Step 1 Load `src\CyberSource.Clients.sln` in Visual Studio 2005.

Step 2 Enter the URL of the new WSDL file in the Web Reference URL of the SoapWebReference web reference. This can be done automatically by right-clicking SoapWebReference and selecting `Update Web Reference`. After a successful update, a new `Reference.cs` file is generated in the `Web References\SoapWebReference` directory.

Step 3 Load the newly-generated `Reference.cs` and look for the constructor for TransactionProcessWse.

Step 4 Update the constructor so that it accepts the parameter `string url` and replace the hard-coded URL assigned to `this.URL` with this parameter. For example:

```
public TransactionProcessorWse(string url) {

this.Url = url;
```

Step 5 In the Project Properties, switch to the Signing tab and choose your own strong name key file or uncheck the `Sign the assembly` check box if you do not want to sign the assembly. To generate your own key file, use the Strong Name tool (`sn.exe`) that is included with the .NET Framework SDK.

Step 6 Build the Release configuration. If this results in build errors related to TransactionProcessorWse, do the following:

a Rerun the WSE 3.0 installer. You can do this by using Add/Remove Programs.

b In the installer, select `modify` and install the `Visual Studio Tools` option.

c Restart Visual Studio.

d Return to step 1.

Step 7 Save a copy of the original `CyberSource.Clients.dll` and then replace it with the newly built `CyberSource.Clients.dll`.

XML Client

Updating the client is unnecessary. Start using the new namespace URI in your input XML documents. The client automatically uses the specified version.

Using Name-Value Pairs

This section explains how to request CyberSource services by using name-value pairs.

Requesting CyberSource Services

To request CyberSource services, write code that:

- Collects information for the CyberSource services that you will use
- Assembles the order information into requests
- Sends the requests to the CyberSource server
- Processes the reply information

Important The CyberSource servers do not support persistent HTTP connections.

The instructions in this section explain how to write C# programs that request CyberSource services. For a list of API fields to use in your requests, see "Related Documents," page 21.

Creating and Sending the Request

To use any CyberSource service, you must create and send a request that includes the required information for that service.

The example developed in the following sections shows basic code for requesting CyberSource services. In this example, Jane Smith is buying an item for $29.95.

Important The code in this section's examples is incomplete. For complete sample programs, see the source code in the client's samples\src\nvp directory.

Creating a New Visual Studio .NET Project

To get started, create a new project in Visual Studio .NET, and add a reference to the client library, `CyberSource.Clients.dll`, which is located in the client's lib directory. You must also add a reference to the library `System.Web.Services.dll`, which is part of the .NET Framework.

Importing the Client Classes

In the code for your application, add the following import statements:

```
using CyberSource.Clients;
using System;
using System.Collections;
using System.Configuration;
using System.Net;
using System.Web.Services.Protocols;
```

Creating an Empty Request

You next create a hashtable that holds the request fields:

```
Hashtable request = new Hashtable();
```

Adding the Merchant ID

You next optionally add your CyberSource merchant ID to the request:

```
request.Add( "merchantID", "infodev" );
```

This value overrides any value you set with the merchantID configuration setting (see Table 34, "Fields in the Settings File," on page 171). The merchant ID is case sensitive.

Adding Services to the Request

You next indicate the service that you want to use by adding a field to the request. For example, to request a credit card authorization:

```
request.Add( "ccAuthService_run", "true" );
```

Requesting a Sale

You can request multiple services by adding additional fields to the request. For example, if you fulfill the order immediately, you can request a credit card authorization and capture together (also referred to as a “sale”):

```
request.Add( "ccAuthService_run", "true" );
request.Add( "ccCaptureService_run", "true" );
```

Adding Service-Specific Fields to the Request

You next add the fields that are used by the services you are requesting. If you request multiple services and they share common fields, you must add the field once only.

```
request.Add( "billTo_firstName", "Jane" );
request.Add( "billTo_lastName", "Smith" );
request.Add( "card_accountNumber", "4111111111111111" );
request.Add( "item_0_unitPrice", "29.95" );
```

The previous example shows only a partial list of the fields you must send. Refer to "Requesting CyberSource Services," page 178 for information about the guides that list all of the fields for the services that you are requesting.

Sending the Request

You next send the request to CyberSource, store the reply in a new hash table, and catch several exceptions that you might receive:

```
try {
  Hashtable reply = NVPClient.RunTransaction( request );
  SaveOrderState();
  // "Using the Decision and Reason Code," page 183 describes the ProcessReply
  // method.
  ProcessReply( reply );
} catch (SignException se) {
  SaveOrderState();
  Console.WriteLine( se.ToString() );
} catch (SoapHeaderException she) {
  SaveOrderState();
  Console.WriteLine( she.ToString() );
} catch (SoapBodyException sbe) {
  SaveOrderState();
  /*
   * Some types of SoapBodyException indicate that the transaction may have been
   * completed by CyberSource. The sample code shows how to identify these
   * excceptions. If you receive such an exception, and your request included a
   * payment service, you should use the CyberSource transaction search screens to
   * determine whether the transaction was processed.
   */
  Console.WriteLine( sbe.ToString() );
} catch (WebException we) {
  SaveOrderState();
  /*
   * Some types of WebException indicate that the transaction may have been
   * completed by CyberSource. The sample code shows how to identify these
   * exceptions. If you receive such an exception, and your request included a
   * payment service, you should use the CyberSource transaction search screens to
   * determine whether the transaction was processed.
   */
  Console.WriteLine( we.ToString() );
}private static void SaveOrderState() {
  /*

   * This is where you store the order state in your system for post-transaction
   * analysis. Be sure to store the consumer information, the values of the reply
   * fields, and the details of any exceptions that occurred.
   */
}
```

In the preceding example, when an exception occurs, the exception is printed to the console. Your web store should also display a message to the consumer indicating that you were unable to process the order. The sample code for the name-value pair client shows you how to provide feedback to the consumer.

Also, if the transaction fails, and the request did not include any payment services, you may be able to resend the transaction. The sample code for the name-value pair client shows you how to do this.

Interpreting the Reply

After the CyberSource server processes your request, it sends a reply message that contains information about the services you requested. You receive different fields depending on the services you request and the outcome of each service.

To use the reply information, you must integrate it into your system and any other system that uses that data. For example, you can store the reply information in a database and send it to other back office applications.

You must write an error handler to process the reply information that you receive from CyberSource. Do not show the reply information directly to consumers. Instead, present an appropriate response that tells consumers the result.

Important Because CyberSource may add reply fields and reason codes at any time, you should parse the reply data according to the names of the fields instead of their order in the reply.

The most important reply fields to evaluate are the following:

- **decision**: A one-word description of the results of your request. The decision is one of the following:
  - `ACCEPT` if the request succeeded
  - `REJECT` if one or more of the services in the request was declined
  - `REVIEW` if you use CyberSource Decision Manager and it flags the order for review. See "For CyberSource Advanced Merchants: Handling Decision Manager Reviews," page 185 for more information.
  - `ERROR` if there was a system error. See "Retrying When System Errors Occur," page 187 for important information about handling retries in the case of system errors.
- **reasonCode**: A numeric code that provides more specific information about the results of your request.

You also receive a reason code for each service in your request. You can use these reason codes to determine whether a specific service succeeded or failed. If a service fails, other services in your request may not run. For example, if you request a credit card

authorization and capture, and the authorization fails, the capture does not run. The reason codes for each service are described in the *Credit Card Services User Guide* for CyberSource Essentials merchants or in the service developer guide for CyberSource Advanced merchants.

Important CyberSource reserves the right to add new reason codes at any time. If your error handler receives a reason code that it does not recognize, it should use the decision to interpret the reply.

Using the Decision and Reason Code

The following example shows how you can use the decision and the reason code to display an appropriate message to the consumer.

```
private static bool ProcessReply( Hashtable reply ) {
  string template = GetTemplate(
    ((string)reply["decision"]).ToUpper() );
  string content  = GetContent( reply );

  // This example writes the message to the console. Choose an appropriate display
  // method for your own application.
  Console.WriteLine( template, content );
}

private static string GetTemplate( string decision ) {
  // Retrieves the text that corresponds to the decision.

  if ("ACCEPT".Equals( decision )) {
    return( "The order succeeded.{0}" );
  }

  if ("REJECT".Equals( decision )) {
    return( "Your order was not approved.{0}" );
  }

  // ERROR, or an unknown decision
  return( "Your order could not be completed at this time.{0}" +
          "\nPlease try again later." );
}
private static string GetContent( Hashtable reply ) {
  /*
   * Uses the reason code to retrieve more details to add to the template.
   *
   * The messages returned in this example are meant to demonstrate how to
   * retrieve the reply fields. Your application should display user-friendly
   * messages.
   */
  int reasonCode = int.Parse( (string) reply["reasonCode"] );
  switch (reasonCode) {
    // Success
    case 100:
      return( "\nRequest ID: " + reply["requestID"] );

    // Missing field or fields

    case 101:
      return( "\nThe following required fields are missing: " +
              EnumerateValues( reply, "missingField" ) );

    // Invalid field or fields
    case 102:
      return( "\nThe following fields are invalid: " +
              EnumerateValues( reply, "invalidField" ) );

    // Insufficient funds
    case 204:
      return( "\nInsufficient funds in the account. Please use a " +
              "different card or select another form of payment." );

    // Add additional reason codes here that you must handle more specifically.

    default:
      // For all other reason codes, such as unrecognized reason codes, or codes
      // that do not require special handling, return an empty string.
      return( String.Empty );
  }
}
private static string EnumerateValues( Hashtable reply,
                                       string fieldName ) {
  System.Text.StringBuilder sb = new System.Text.StringBuilder();
  string val = "";
  for (int i = 0; val != null; ++i) {
    val = (string) reply[fieldName + "_" + i];
    if (val != null) {
      sb.Append( val + "\n" );
    }
  }

  return( sb.ToString() );
}
```

For CyberSource Advanced Merchants: Handling Decision Manager Reviews

The information in this section applies only to CyberSource Advanced merchants.

If you use CyberSource Decision Manager, you may also receive the `REVIEW` value in the **decision** field. `REVIEW` means that Decision Manager has marked the order for review based on how you configured the Decision Manager rules.

If you will be using Decision Manager, you have to determine how to handle the new `REVIEW` value. Ideally, you will update your order management system to recognize the `REVIEW` response and handle it according to your business rules. If you cannot update your system to handle the `REVIEW` response, CyberSource recommends that you choose one of these options:

- If you authorize and capture the credit card payment at the same time, treat the `REVIEW` response like a `REJECT` response. Rejecting any orders that are marked for review may be appropriate if your product is a software download or access to a Web site. If supported by your processor, you may also want to reverse the authorization.

- If you approve the order after reviewing it, convert the order status to `ACCEPT` in your order management system. You can request the credit card capture without requesting a new authorization.

- If you approve the order after reviewing it but cannot convert the order status to `ACCEPT` in your system, request a new authorization for the order. When processing this new authorization, you must disable Decision Manager. Otherwise the order will be marked for review again. For details about the API field that disables Decision Manager, see the *Decision Manager Developer Guide Using the Simple Order API* (PDF | HTML) or the *Decision Manager Developer Guide Using the SCMP Order API* (PDF | HTML).

Alternately, you can specify a custom business rule in Decision Manager so that authorizations originating from a particular internal IP address at your company are automatically accepted.

If supported by your processor, you may want to reverse the original authorization.

Requesting Multiple Services

When you request multiple services in one request, CyberSource processes the services in a specific order. If a service fails, CyberSource does not process the subsequent services in the request.

For example, in the case of a sale (a credit card authorization and a capture requested together), if the authorization service fails, CyberSource will not process the capture service. The reply you receive only includes reply fields for the authorization.

Many CyberSource services include “ignore” fields that tell CyberSource to ignore the result from the first service when deciding whether to run the subsequent services. In the case of the sale, even though the issuing bank gives you an authorization code, CyberSource might decline the authorization based on the AVS or card verification results. Depending on your business needs, you might choose to capture these types of declined authorizations anyway. You can set the **businessRules_ignoreAVSResult** field to `"true"` in your combined authorization and capture request:

```
request.put( "businessRules_ignoreAVSResult", "true" );
```

This tells CyberSource to continue processing the capture even if the AVS result causes CyberSource to decline the authorization. In this case you would then get reply fields for both the authorization and the capture in your reply.

Note

You are charged only for the services that CyberSource performs.

Retrying When System Errors Occur

You must design your transaction management system to include a way to correctly handle CyberSource system errors. Depending on which payment processor is handling the transaction, the error may indicate a valid CyberSource system error, or it may indicate a processor rejection because of some type of invalid data. In either case, CyberSource recommends that you do not design your system to retry sending a transaction many times in the case of a system error.

Instead, CyberSource recommends that you retry sending the request only two or three times with successively longer periods of time between each retry. For example, after the first system error response, wait 30 seconds and then retry sending the request. If you receive the same error a second time, wait one minute before you send the request again. Depending on the situation, you may decide you can retry sending the request after a longer time period. Determine what is most appropriate for your business situation.

If after several retry attempts you are still receiving a system error, it is possible that the error is actually being caused by a processor rejection and not a CyberSource system error. In that case, CyberSource suggest that you either:

- Search for the transaction in the Business Center (depending on which one you normally use), look at the description of the error on the Transaction Detail page, and call your processor to determine if and why they are rejecting the transaction.
- Contact CyberSource Customer Support to confirm whether your error is truly caused by a CyberSource system issue.

If TSYS Acquiring Solutions is your processor, you may want to follow the first suggestion as there are several common TSYS Acquiring Solutions processor responses that are returned to you as system errors and that only TSYS Acquiring Solutions can address.

Creating an Application Settings File

After you finish writing code for your integration, you must create an application settings file. This file must contain at least the following information:

- The directory that contains your security key
- The location of the CyberSource server

See Table 34, "Fields in the Settings File," on page 171 for a complete list of settings.

You can use the settings files that come with the sample applications as a starting point for your own settings file. See "Configuring the Test Applications," page 170 for more information.

Using XML

This section explains how to request CyberSource services by using XML.

Requesting CyberSource Services

To request CyberSource services, write code that:

- Collects information for the CyberSource services that you will use
- Assembles the order information into requests
- Sends the requests to the CyberSource server
- Processes the reply information

Important The CyberSource servers do not support persistent HTTP connections.

The instructions in this section explain how to write C# programs that request CyberSource services. For a list of API fields to use in your requests, see "Related Documents," page 21.

Creating a Request Document

The XML client allows you to create an XML request document using any application, then send the request to CyberSource. For example, if you have a customer relationship management (CRM) system that uses XML to communicate with other systems, you can use the CRM system to generate request documents.

The request document must validate against the XML schema for CyberSource transactions. To view the schema, go to

https://ics2wsa.ic3.com/commerce/1.x/transactionProcessor

and look at the `xsd` file for the version of the Simple Order API you are using.

Important

Make sure that the elements in your document appear in the correct order. If they do not, your document will not validate, and your request will fail.

The following example shows a basic XML document for requesting CyberSource services. In this example, Jane Smith is buying an item for $29.95.

Note

The XML document in this example is incomplete. For complete examples, see `sample.xml` in the client's `samples\bin` directory.

Creating an Empty Request

Add the XML declaration and the document's root element:

```
<?xml version="1.0" encoding="utf-8"?>
<requestMessage xmlns="urn:schemas-cybersource-com:transaction-data-
1.17">
</requestMessage>
```

Make sure that the API version specified at the end of the namespace is correct. For example, to communicate with version 1.19, you must use the namespace `urn:schemas-cybersource-com:transaction-data-1.19`. When you must update the API version, see "Updating the Client to Use a Later API Version," page 176.

Note

The XML document that you receive in the reply always has the prefix `c:`, for example: `xmlns:c="urn:schemas-cybersource-com:transaction-data-1.17"`. Make sure you use an XML parser that supports namespaces.

Adding the Merchant ID

Optionally, you can add the CyberSource merchant ID to the request:

```
<?xml version="1.0" encoding="utf-8"?>
<requestMessage xmlns="urn:schemas-cybersource-com:transaction-data-
1.17">
  <merchantID>infodev</merchantID>
</requestMessage>
```

This value overrides any value that you set with the merchantID configuration setting. For more information about the merchantID configuration setting, see Table 34, "Fields in the Settings File," on page 171. The merchant ID is case sensitive.

Adding Services to the Request

You next indicate the service that you want to use by creating an element for that service in the request, then setting the element's `run` attribute to `true`. For example, to request a credit card authorization:

```
<?xml version="1.0" encoding="utf-8"?>
<requestMessage xmlns="urn:schemas-cybersource-com:transaction-data-
1.15">
  <merchantID>infodev</merchantID>
  <ccAuthService run="true"/>
</requestMessage>
```

Requesting a Sale

You can request multiple services by creating additional elements. For example, if you fulfill the order immediately, you can request a credit card authorization and capture together (referred to as a "sale"):

```
<?xml version="1.0" encoding="utf-8"?>
<requestMessage xmlns="urn:schemas-cybersource-com:transaction-data-
1.17">
  <merchantID>infodev</merchantID>
  <ccAuthService run="true"/>
  <ccCaptureService run="true"/>
</requestMessage>
```

Adding Service-Specific Fields to the Request

You next add the fields that are used by the services you are requesting. Most fields are child elements of container elements; for example, a `<card>` element contains the consumer's credit card information.

```
<?xml version="1.0" encoding="utf-8"?>
<requestMessage xmlns="urn:schemas-cybersource-com:transaction-data-
1.15">
  <merchantID>infodev</merchantID>
  <billTo>
    <firstName>Jane</firstName>
    <lastName>Smith</lastName>
  </billTo>
  <item id="0">
  </card>
    <unitPrice>29.95</unitPrice>
  </item>
  <card>
    <accountNumber>4111111111111111</accountNumber>
  <ccAuthService run="true"/>
</requestMessage>
```

The example above shows only a partial list of the fields you must send. Refer to "Related Documents," page 21 for information about the guides that list all of the fields for the services that you are requesting.

Sending the Request

Once you have created an XML request document, you can use a .NET application to send the request to CyberSource. The example that follows is written in C#.

Note The code in this section's examples is incomplete. For complete sample programs, see the source code in the client's `samples\src\xml` directory.

Creating a New Visual Studio .NET Project

To start, create a new project in Visual Studio .NET, and add a reference to the client library, `CyberSource.Clients.dll`, which is located in the client's lib directory.

Importing the Client Classes

In the code for your application, add the following import statements:

```
using CyberSource.Clients;
using System;
using System.Net;
using System.Xml;
```

Sending the Request

You next read the XML request document, send the request to CyberSource, store the reply in a new `XmlDocument` object, and catch several exceptions that you might receive:

```
try {
  XmlDocument request = new XmlDocument();
  request.Load( "MyXmlDocument.xml" );

  XmlDocument reply = XmlClient.RunTransaction( request );
  SaveOrderState();
  // "Using the Decision and Reason Code," page 183 describes the ProcessReply
  // method.
  ProcessReply( reply );
} catch (SignException se) {
  SaveOrderState();
  Console.WriteLine( se.ToString() );
} catch (FaultException fe) {
  SaveOrderState();
  /*
   * Some types of FaultException indicate that the transaction may have been
   * completed by CyberSource. The sample code shows how to identify these
   * exceptions. If you receive such an exception, and your request included a
   * payment service, you should use the CyberSource transaction search screens to
   * determine whether the transaction was processed.
   */
  Console.WriteLine( fe.ToString() );
} catch (WebException we) {
  SaveOrderState();
  /*
   * Some types of WebException indicate that the transaction may have been completed
   * by CyberSource. The sample code shows how to identify these exceptions. If you
   * receive such an exception, and your request included a payment service, you
   * should use the CyberSource transaction search screens to determine whether the
   * transaction was processed.
   */
  Console.WriteLine( we.ToString() );
}
```

```
private static void SaveOrderState() {
   /*
    * This is where you store the order state in your system for post-transaction
    * analysis. Be sure to store the consumer information, the values of the reply
    * fields, and the details of any exceptions that occurred.
    */
}
```

In the preceding example, when an exception occurs, the exception is printed to the console. Your web store should also display a message to the consumer indicating that you were unable to process the order. The sample code for the XML client shows you how to provide feedback to the consumer.

Also, if the transaction fails, and the request did not include any payment services, you may be able to resend the transaction. The sample code for the XML client shows you how to do this.

Interpreting the Reply

After the CyberSource server processes your request, it sends a reply message that contains information about the services you requested. You receive different fields depending on the services you request and the outcome of each service.

To use the reply information, you must integrate it into your system and any other system that uses that data. For example, you can store the reply information in a database and send it to other back office applications.

You must write an error handler to process the reply information that you receive from CyberSource. Do not show the reply information directly to consumers. Instead, present an appropriate response that tells consumers the result.

Important

Because CyberSource may add reply fields and reason codes at any time, you should parse the reply data according to the names of the fields instead of their order in the reply.

The most important reply fields to evaluate are the following:

- **decision**: A one-word description of the results of your request. The decision is one of the following:
  - `ACCEPT` if the request succeeded
  - `REJECT` if one or more of the services in the request was declined
  - `REVIEW` if you use CyberSource Decision Manager and it flags the order for review. See "For CyberSource Advanced Merchants: Handling Decision Manager Reviews," page 197 for more information.
  - `ERROR` if there was a system error. See "Retrying When System Errors Occur," page 199 for important information about handling retries in the case of system errors.
- **reasonCode**: A numeric code that provides more specific information about the results of your request.

You also receive a reason code for each service in your request. You can use these reason codes to determine whether a specific service succeeded or failed. If a service fails, other services in your request may not run. For example, if you request a credit card authorization and capture, and the authorization fails, the capture does not run. The reason codes for each service are described in the *Credit Card Services User Guide* for CyberSource Essentials merchants or in the service developer guide for CyberSource Advanced merchants.

Important CyberSource reserves the right to add new reason codes at any time. If your error handler receives a reason code that it does not recognize, it should use the decision to interpret the reply.

Using the Decision and Reason Code

The following example shows how you can use the decision and the reason code to display an appropriate message to the consumer.

```
private static bool ProcessReply( XmlDocument reply ) {
  // The following code allows you to use XPath with the CyberSource schema, which
  // uses a non-empty default namespace.
  XmlNamespaceManager nsmgr
    = new XmlNamespaceManager( reply.NameTable );
  nsmgr.AddNamespace( "cybs", Client.CYBS_NAMESPACE );

  XmlNode replyMessage
    = reply.SelectSingleNode( "cybs:replyMessage", nsmgr);

  string decision = replyMessage.SelectSingleNode(
    "cybs:decision/text()", nsmgr ).Value;

  string template = GetTemplate( decision.ToUpper() );
  string content  = GetContent( replyMessage, nsmgr );

  // This example writes the message to the console. Choose an appropriate display
  // method for your own application.
  Console.WriteLine( template, content );
}
private static string GetTemplate( string decision ) {
  // Retrieves the text that corresponds to the decision.
  if ("ACCEPT".Equals( decision )) {
    return( "The order succeeded.{0}" );
  }

}
  if ("REJECT".Equals( decision )) {
    return( "Your order was not approved.{0}" );
  }

  // ERROR, or an unknown decision
  return( "Your order could not be completed at this time.{0}" +
          "\nPlease try again later." );
private static string GetContent(
  XmlNode replyMessage, XmlNamespaceManager nsmgr ) {
  /*
   * Uses the reason code to retrieve more details to add to the template.
   *
   * The messages returned in this example are meant to demonstrate how to retrieve
   * the reply fields. Your application should display user-friendly messages.
   */
  string textVal = replyMessage.SelectSingleNode(
    "cybs:reasonCode/text()", nsmgr ).Value;
  int reasonCode = int.Parse( textVal );
  switch (reasonCode) {
    // Success
    case 100:
      return( "\nRequest ID: " +
        replyMessage.SelectSingleNode(
          "cybs:requestID/text()", nsmgr ).Value );
    // Missing field or fields
    case 101:
      return( "\nThe following required fields are missing: " +
              EnumerateValues( replyMessage.SelectNodes(
                "cybs:missingField/text()", nsmgr ) ) );

    // Invalid field or fields
    case 102:
      return( "\nThe following fields are invalid: " +
              EnumerateValues( replyMessage.SelectNodes(
                "cybs:invalidField/text()", nsmgr ) ) );
    // Insufficient funds
    case 204:
      return( "\nInsufficient funds in the account. Please use a " +
              "different card or select another form of payment." );

    // Add additional reason codes here that you must handle more specifically.

    default:
      // For all other reason codes (for example, unrecognized reason codes, or
      // codes that do not require special handling), return an empty string.
      return( String.Empty );
  }
}
private static string EnumerateValues( XmlNodeList nodes ) {
  System.Text.StringBuilder sb = new System.Text.StringBuilder();
  foreach (XmlNode node in nodes) {
    sb.Append( val + "\n" );
  }
  return( sb.ToString() );
}
```

For CyberSource Advanced Merchants: Handling Decision Manager Reviews

The information in this section applies only to CyberSource Advanced merchants.

If you use CyberSource Decision Manager, you may also receive the `REVIEW` value in the **decision** field. `REVIEW` means that Decision Manager has marked the order for review based on how you configured the Decision Manager rules.

If you will be using Decision Manager, you have to determine how to handle the new `REVIEW` value. Ideally, you will update your order management system to recognize the `REVIEW` response and handle it according to your business rules. If you cannot update your system to handle the `REVIEW` response, CyberSource recommends that you choose one of these options:

- If you authorize and capture the credit card payment at the same time, treat the `REVIEW` response like a `REJECT` response. Rejecting any orders that are marked for review may be appropriate if your product is a software download or access to a Web site. If supported by your processor, you may also want to reverse the authorization.

- If you approve the order after reviewing it, convert the order status to `ACCEPT` in your order management system. You can request the credit card capture without requesting a new authorization.

- If you approve the order after reviewing it but cannot convert the order status to `ACCEPT` in your system, request a new authorization for the order. When processing this new authorization, you must disable Decision Manager. Otherwise the order will be marked for review again. For details about the API field that disables Decision Manager, see the *Decision Manager Developer Guide Using the Simple Order API* (PDF | HTML) or the *Decision Manager Developer Guide Using the SCMP Order API* (PDF | HTML).

Alternately, you can specify a custom business rule in Decision Manager so that authorizations originating from a particular internal IP address at your company are automatically accepted.

If supported by your processor, you may want to reverse the original authorization.

Requesting Multiple Services

When you request multiple services in one request, CyberSource processes the services in a specific order. If a service fails, CyberSource does not process the subsequent services in the request.

For example, in the case of a sale (a credit card authorization and a capture requested together), if the authorization service fails, CyberSource will not process the capture service. The reply you receive only includes reply fields for the authorization.

Many CyberSource services include “ignore” fields that tell CyberSource to ignore the result from the first service when deciding whether to run the subsequent services. In the case of the sale, even though the issuing bank gives you an authorization code, CyberSource might decline the authorization based on the AVS or card verification results. Depending on your business needs, you might choose to capture these types of declined authorizations anyway. You can set the **businessRules_ignoreAVSResult** field to `"true"` in your combined authorization and capture request:

```
<businessRules>
    <ignoreAVSResult>true</ignoreAVSResult>
</businessRules>
```

This tells CyberSource to continue processing the capture even if the AVS result causes CyberSource to decline the authorization. In this case you would then get reply fields for both the authorization and the capture in your reply.

Note You are charged only for the services that CyberSource performs.

Retrying When System Errors Occur

You must design your transaction management system to include a way to correctly handle CyberSource system errors. Depending on which payment processor is handling the transaction, the error may indicate a valid CyberSource system error, or it may indicate a processor rejection because of some type of invalid data. In either case, CyberSource recommends that you do not design your system to retry sending a transaction many times in the case of a system error.

Instead, CyberSource recommends that you retry sending the request only two or three times with successively longer periods of time between each retry. For example, after the first system error response, wait 30 seconds and then retry sending the request. If you receive the same error a second time, wait one minute before you send the request again. Depending on the situation, you may decide you can retry sending the request after a longer time period. Determine what is most appropriate for your business situation.

If after several retry attempts you are still receiving a system error, it is possible that the error is actually being caused by a processor rejection and not a CyberSource system error. In that case, we suggest that you either:

- Search for the transaction in the Business Center (depending on which one you normally use), look at the description of the error on the Transaction Detail page, and call your processor to determine if and why they are rejecting the transaction.

- Contact CyberSource Customer Support to confirm whether your error is truly caused by a CyberSource system issue.

If TSYS Acquiring Solutions is your processor, you may want to follow the first suggestion as there are several common TSYS Acquiring Solutions processor responses that are returned to you as system errors and that only TSYS Acquiring Solutions can address.

Creating an Application Settings File

After you finish writing code for your integration, you must create an application settings file. This file must contain, at a minimum, the following information:

- The directory that contains your security key
- The location of the CyberSource server

See Table 34, "Fields in the Settings File," on page 171 for a complete list of settings.

You can use the settings files that come with the sample applications as a starting point for your own settings file. See "Configuring the Test Applications," page 170 for more information.

Using SOAP

This section explains how to request CyberSource services by using the Simple Object Access Protocol (SOAP).

Requesting CyberSource Services

To request CyberSource services, write code that:

- Collects information for the CyberSource services that you will use
- Assembles the order information into requests
- Sends the requests to the CyberSource server

> **Important** The CyberSource servers do not support persistent HTTP connections.

- Processes the reply information

The instructions in this section explain how to write C# programs that request CyberSource services. For a list of API fields to use in your requests, see "Related Documents," page 21.

Creating and Sending the Request

To use any CyberSource service, you must create and send a request that includes the required information for that service.

The following example shows basic code for requesting CyberSource services. In this example, Jane Smith is buying an item for $29.95.

> **Important** The code in this section's examples is incomplete. For complete sample programs, see the source code in the client's `samples\src\soap` directory.

Creating a New Visual Studio .NET Project

To get started, create a new project in Visual Studio .NET, and add a reference to the client library, `CyberSource.Clients.dll`, which is located in the client's lib directory. You must also add a reference to the library `System.Web.Services.dll`, which is part of the .NET Framework.

Importing the Client Classes

In the code for your application, add the following import statements:

```
using CyberSource.Clients.dll;
using CyberSource.Clients.SoapWebReference;
using System;
using System.Configuration;
using System.Net;
using System.Web.Services.Protocols;
```

Creating an Empty Request

You next create a RequestMessage object that holds the request fields:

```
RequestMessage request = new RequestMessage();
```

Adding the Merchant ID

You next optionally add your CyberSource merchant ID to the request:

```
request.merchantID = "infodev";
```

This value overrides any value you set with the merchantID configuration setting (see Table 34, "Fields in the Settings File," on page 171). The merchant ID is case sensitive.

Adding Services to the Request

You next indicate the service that you want to use by creating an object for that service in the request, then setting the object's `run` property to `true`. For example, to request a credit card authorization:

```
request.ccAuthService = new CCAuthService();
request.ccAuthService.run = "true";
```

Requesting a Sale

You can request multiple services by creating additional objects. For example, if you fulfill the order immediately, you can request a credit card authorization and capture together (referred to as a “sale”):

```
request.ccAuthService = new CCAuthService();
request.ccAuthService.run = "true";
request.ccCaptureService = new CCCaptureService();
request.ccCaptureService.run = "true";
```

Adding Service-Specific Fields to the Request

You next add the fields that are used by the services you are requesting. Most fields are properties of additional objects; for example, a `Card` object contains the consumer’s credit card information.

```
BillTo billTo = new BillTo();
billTo.firstName = "Jane";
billTo.lastName = "Smith";
request.billTo = billTo;

Card card = new Card();
card.accountNumber = "4111111111111111";
request.card = card;

// there is one item in this sample
request.item = new Item[1];
Item item = new Item();
item.id = "0";
item.unitPrice = "29.95";
request.item[0] = item;
```

The example above shows only a partial list of the fields you must send. Refer to "Related Documents," page 21 for information about the guides that list all of the fields for the services that you are requesting.

Sending the Request

You next send the request to CyberSource, store the reply in a new `ReplyMessage` object, and handle several exceptions that you might receive.

```
try {
  ReplyMessage reply = SoapClient.RunTransaction( request );
  SaveOrderState();
  // "Using the Decision and Reason Code," page 183 describes the ProcessReply
  // method.
  ProcessReply( reply );
} catch (SignException se) {
  SaveOrderState();
  Console.WriteLine( se.ToString() );
} catch (SoapHeaderException she) {
  SaveOrderState();
  Console.WriteLine( she.ToString() );
} catch (SoapBodyException sbe) {
  SaveOrderState();
  /*
   * Some types of SoapBodyException indicate that the transaction may have been
   * completed by CyberSource. The sample code shows how to identify these exceptions.
   * If you receive such an exception, and your request included a payment service,
   * you should use the CyberSource transaction search screens to determine whether
   * the transaction was processed.
   */
  Console.WriteLine( sbe.ToString() );
} catch (WebException we) {
  SaveOrderState();
  /*
   * Some types of WebException indicate that the transaction may have been
   * completed by CyberSource. The sample code shows how to identify these exceptions.
   * If you receive such an exception, and your request included a payment service,
   * you should use the CyberSource transaction search screens to determine whether
   * the transaction was processed.
   */
  Console.WriteLine( we.ToString() );
}
private static void SaveOrderState() {
  /*
   * This is where you store the order state in your system for post-transaction
   * analysis. Be sure to store the consumer information, the values of the reply
   * fields, and the details of any exceptions that occurred.
   */
}
```

In the preceding example, when an exception occurs, the exception is printed to the console. Your web store should also display a message to the consumer indicating that you were unable to process the order. The sample code for the SOAP client shows you how to provide feedback to the consumer.

Also, if the transaction fails, and the request did not include any payment services, you may be able to resend the transaction. The sample code for the SOAP client shows you how to do this.

Interpreting the Reply

After the CyberSource server processes your request, it sends a reply message that contains information about the services you requested. You receive different fields depending on the services you request and the outcome of each service.

To use the reply information, you must integrate it into your system and any other system that uses that data. For example, you can store the reply information in a database and send it to other back office applications.

You must write an error handler to process the reply information that you receive from CyberSource. Do not show the reply information directly to consumers. Instead, present an appropriate response that tells consumers the result.

⚠ **Important**

Because CyberSource may add reply fields and reason codes at any time, you should parse the reply data according to the names of the fields instead of their order in the reply.

The most important reply fields to evaluate are the following:

- **decision**: A one-word description of the results of your request. The decision is one of the following:
  - `ACCEPT` if the request succeeded
  - `REJECT` if one or more of the services in the request was declined
  - `REVIEW` if you use CyberSource Decision Manager and it flags the order for review. See "For CyberSource Advanced Merchants: Handling Decision Manager Reviews," page 207 for more information.
  - `ERROR` if there was a system error. See "Retrying When System Errors Occur," page 209 for important information about handling retries in the case of system errors.
- **reasonCode**: A numeric code that provides more specific information about the results of your request.

You also receive a reason code for each service in your request. You can use these reason codes to determine whether a specific service succeeded or failed. If a service fails, other services in your request may not run. For example, if you request a credit card

authorization and capture, and the authorization fails, the capture does not run. The reason codes for each service are described in the *Credit Card Services User Guide* for CyberSource Essentials merchants or in the service developer guide for CyberSource Advanced merchants.

Important CyberSource reserves the right to add new reason codes at any time. If your error handler receives a reason code that it does not recognize, it should use the decision to interpret the reply.

Using the Decision and Reason Code

The following example shows how you can use the decision and the reason code to display an appropriate message to the consumer.

```
private static bool ProcessReply( ReplyMessage reply ) {
  string template = GetTemplate( reply.decision.ToUpper() );
  string content  = GetContent( reply );

  // This example writes the message to the console. Choose an appropriate display
  // method for your own application.
  Console.WriteLine( template, content );
}

private static string GetTemplate( string decision ) {
  // Retrieves the text that corresponds to the decision.
  if ("ACCEPT".Equals( decision )) {
    return( "The order succeeded.{0}" );
  }
  if ("REJECT".Equals( decision )) {
    return( "Your order was not approved.{0}" );

  }

  // ERROR, or an unknown decision
  return( "Your order could not be completed at this time.{0}" +
          "\nPlease try again later." );
}
private static string GetContent( ReplyMessage reply ) {
  /*
   * Uses the reason code to retrieve more details to add to the template.
   * The messages returned in this example are meant to demonstrate how to retrieve
   * the reply fields. Your application should display user-friendly messages.
   */
  int reasonCode = int.Parse( reply.reasonCode );
  switch (reasonCode) {
    // Success
    case 100:
      return( "\nRequest ID: " + reply.requestID );
    // Missing field or fields
    case 101:
      return( "\nThe following required fields are missing: " +
              EnumerateValues( reply.missingField ) );

    // Invalid field or fields

    case 102:
      return( "\nThe following fields are invalid: " +
              EnumerateValues( reply.invalidField ) );

    // Insufficient funds
    case 204:
      return( "\nInsufficient funds in the account. Please use a " +
              "different card or select another form of payment." );
    // Add additional reason codes here that you must handle more specifically.
    default:
      // For all other reason codes, such as unrecognized reason codes or codes
      // that do not require special handling, return an empty string.
      return( String.Empty );
  }
}
private static string EnumerateValues( string[] array ) {
  System.Text.StringBuilder sb = new System.Text.StringBuilder();
  foreach (string val in array) {
    sb.Append( val + "\n" );

  }
  return( sb.ToString() );
}
```

For CyberSource Advanced Merchants: Handling Decision Manager Reviews

The information in this section applies only to CyberSource Advanced merchants.

If you use CyberSource Decision Manager, you may also receive the `REVIEW` value in the **decision** field. `REVIEW` means that Decision Manager has marked the order for review based on how you configured the Decision Manager rules.

If you will be using Decision Manager, you have to determine how to handle the new `REVIEW` value. Ideally, you will update your order management system to recognize the `REVIEW` response and handle it according to your business rules. If you cannot update your system to handle the `REVIEW` response, CyberSource recommends that you choose one of these options:

- If you authorize and capture the credit card payment at the same time, treat the `REVIEW` response like a `REJECT` response. Rejecting any orders that are marked for review may be appropriate if your product is a software download or access to a Web site. If supported by your processor, you may also want to reverse the authorization.

- If you approve the order after reviewing it, convert the order status to `ACCEPT` in your order management system. You can request the credit card capture without requesting a new authorization.

- If you approve the order after reviewing it but cannot convert the order status to `ACCEPT` in your system, request a new authorization for the order. When processing this new authorization, you must disable Decision Manager. Otherwise the order will be marked for review again. For details about the API field that disables Decision Manager, see the *Decision Manager Developer Guide Using the Simple Order API* (PDF | HTML) or the *Decision Manager Developer Guide Using the SCMP Order API* (PDF | HTML).

Alternately, you can specify a custom business rule in Decision Manager so that authorizations originating from a particular internal IP address at your company are automatically accepted.

If supported by your processor, you may want to reverse the original authorization.

Requesting Multiple Services

When you request multiple services in one request, CyberSource processes the services in a specific order. If a service fails, CyberSource does not process the subsequent services in the request.

For example, in the case of a sale (a credit card authorization and a capture requested together), if the authorization service fails, CyberSource will not process the capture service. The reply you receive only includes reply fields for the authorization.

Many CyberSource services include “ignore” fields that tell CyberSource to ignore the result from the first service when deciding whether to run the subsequent services. In the case of the sale, even though the issuing bank gives you an authorization code, CyberSource might decline the authorization based on the AVS or card verification results. Depending on your business needs, you might choose to capture these types of declined authorizations anyway. You can set the **businessRules_ignoreAVSResult** field to `"true"` in your combined authorization and capture request:

```
BusinessRules businessRules = new BusinessRules();

businessRules.ignoreAVSResult = "true";

request.businessRules = businessRules;
```

This tells CyberSource to continue processing the capture even if the AVS result causes CyberSource to decline the authorization. In this case you would then get reply fields for both the authorization and the capture in your reply.

Note You are charged only for the services that CyberSource performs.

Retrying When System Errors Occur

You must design your transaction management system to include a way to correctly handle CyberSource system errors. Depending on which payment processor is handling the transaction, the error may indicate a valid CyberSource system error, or it may indicate a processor rejection because of some type of invalid data. In either case, CyberSource recommends that you do not design your system to retry sending a transaction many times in the case of a system error.

Instead, CyberSource recommends that you retry sending the request only two or three times with successively longer periods of time between each retry. For example, after the first system error response, wait 30 seconds and then retry sending the request. If you receive the same error a second time, wait one minute before you send the request again. Depending on the situation, you may decide you can retry sending the request after a longer time period. Determine what is most appropriate for your business situation.

If after several retry attempts you are still receiving a system error, the error may actually be caused by a processor rejection, not a CyberSource system error. In that case, we suggest one of these actions:

- Search for the transaction in the Business Center (depending on which one you normally use), look at the description of the error on the Transaction Detail page, and call your processor to determine if and why the transaction was rejected.
- Contact CyberSource Customer Support to confirm whether your error is truly caused by a CyberSource system issue.

If TSYS Acquiring Solutions is your processor, you may want to follow the first suggestion because several common TSYS Acquiring Solutions processor responses can be returned as system errors, and only TSYS Acquiring Solutions can address these errors.

Creating an Application Settings File

After you finish writing code for your integration, you must create an application settings file. This file must contain at least the directory that contains your security key and the location of the CyberSource server.

See Table 34, "Fields in the Settings File," on page 171 for a complete list of settings. You can use the settings files that come with the sample applications as a starting point for your own settings file. See "Configuring the Test Applications," page 170 for more information.

Setting the Connection Limit

This section explains how to increase the number of simultaneous connections between the client and CyberSource.

By default, you can create only two simultaneous connections to an HTTP server. By increasing the number of connections, you can avoid a backlog of requests during times of very high transaction volume. Microsoft recommends for the connection limit a value that is 12 times the number of CPUs. For example, if you have two CPUs, you can set the connection limit to 24. To determine the optimum setting for your application, make sure to run performance tests.

Examples

You can increase the number of connections in many ways, for example by using an application- or server-specific configuration file where you can change the setting for a single or for all hosts. The examples below describe briefly some of the methods that you can use to increase connection limits.

cybs.connectionLimit

When set to a value other than `-1`, the `cybs.connectionLimit` setting in the client increases the limit for the host where you are sending the request by executing these statements on your behalf:

```
ServicePoint sp = ServicePointManager.FindServicePoint(uri);
sp.ConnectionLimit = config.ConnectionLimit;
```

<connectionManagement>

You can set the connection limit by using .NET's `<connectionManagement>` tag. In this example, the connection limit for CyberSource's test and production hosts is 12 while the limit for all other hosts is 2:

```
<system.net>
  <connectionManagement>
    <add address = "https://ics2wstesta.ic3.com" maxconnection = "12" />
    <add address = "https://ics2wsa.ic3.com" maxconnection = "12" />
    <add address = "*" maxconnection = "2" />
  </connectionManagement>
</system.net>
```

DefaultConnectionLimit

You can set the connection limit for all hosts to which your application is connected before a connection is made by using the following line in your start-up code:

```
ServicePointManager.DefaultConnectionLimit = your_value_here;
```

References

For more information on these and other methods to increase the connection limits, see the following Microsoft documentation:

- Managing Connections in the *.Net Framework Developer's Guide* (`http://msdn2.microsoft.com/en-us/library/7af54za5.aspx`).

Sample ASP.NET Code Using Visual Basic

The following sample files illustrate how to use the CyberSource Name-Value Pair client in ASP.NET using Visual Basic. The `web.config` file is a sample web application configuration file containing sample entries required by the client. The other files are simple web forms and their corresponding code-behind files. The `Checkout.aspx` file contains a pre-filled form. When you press the Submit button, it will post the entered data to `Checkout2.aspx`, which will send the transaction to CyberSource.

Listing 1: web.config

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

      <add key="cybs.merchantID" value="your_merchant_id"/>
      <add key="cybs.keysDirectory" value="c:\keys"/>
      <add key="cybs.sendToProduction" value="false"/>

      <!-- Logging should normally be disabled in production as it would  -->
      <!-- slow down the processing.  Enable it only when troubleshooting -->
      <!-- an issue.                                                      -->
      <add key="cybs.enableLog" value="false"/>
      <add key="cybs.logDirectory" value="C:\Program Files\CyberSource
Corporation\simapi-net-2.0-5.0.0\logs"/>

      <!-- Please refer to the Connection Limit section in the README for -->
      <!-- details on this setting and alternate ways to set the          -->
      <!-- connection limit.  When not specified or is set to -1, the     -->
      <!-- client will implicitly use the connection limit currently in   -->
      <!-- force, which would be 2 if none of the alternate methods are   -->
      <!-- used.                                                          -->
      <add key="cybs.connectionLimit" value="-1"/>

   </appSettings>
</configuration>
```

Listing 2: Checkout.aspx

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Checkout.aspx.vb"
Inherits="NVP" Debug="true"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
   <title>Name Value Pair - Order Page</title>
</head>
<body>
<form action="Checkout2.aspx" method="post">
Please confirm the information below and click the Submit button to perform the
authorization.
<br/>
   <h3>Billing Information</h3>
   First Name:<br/>
   <input type="text" name="billTo_firstName" value="John"/>
   <br/>Last Name:<br/>
   <input type="text" name="billTo_lastname" value="Doe"/>
   <br/>Street Address:<br/>
   <input type="text" name="billTo_street1" value="1295 Charleston Road"/>
   <br/>City:<br/>
   <input type="text" name="billTo_city" value="Mountain View"/>
   <br/>State:<br/>
   <input type="text" name="billTo_state" value="CA"/>
   <br/>Postal Code:<br/>
   <input type="text" name="billTo_postalCode" value="94043"/>
   <br/>Country:<br/>
   <input type="text" name="billTo_country" value="US"/>
   <br/>
   <h3>Credit Card Information</h3>
   Amount:<br/>
   <input type="text" name="item_0_unitPrice" value="10.00"/>
   <br/>Credit Card Number:<br/>
   <input type="text" name="card_accountNumber" value="4111111111111111"/>
   <br/>Expiration month (mm):<br/>
   <input type="text" name="card_expirationMonth" value="12"/>
   <br/>Expiration year (yyyy):<br/>
   <input type="text" name="card_expirationYear" value="2010"/>
   <br/>Email Address:<br/>
   <input type="text" name="billTo_email" value="nobody@cybersource.com"/>
   <br/><input type="submit" value="Submit"/>
</form>
</body>
</html>
```

Listing 3: Checkout.aspx.vb

```
Partial Class NVP
   Inherits System.Web.UI.Page
End Class
```

Listing 4: Checkout2.aspx

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Checkout2.aspx.vb"
Inherits="NVP2" Debug="true"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
   <title>Name Value Pair - Receipt</title>
</head>
<body>
   <form id="form1" runat="server">
   <div>

   </div>
   </form>
</body>
</html>
```

Listing 5: Checkout2.aspx.vb

```
Imports CyberSource.Clients.NVPClient

Partial Class NVP2
   Inherits System.Web.UI.Page

   Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
      'Declare the request hashtable
      Dim oRequest As New Hashtable

      'Add non-user input fields
      oRequest.Add("ccAuthService_run", "true")
      oRequest.Add("merchantReferenceCode", "MRC-5254555")

      'Add user input fields from post
      oRequest.Add("billTo_firstName", Request.Form("billTo_firstName"))
      oRequest.Add("billTo_lastName", Request.Form("billTo_lastName"))
      oRequest.Add("billTo_street1", Request.Form("billTo_street1"))
      oRequest.Add("billTo_city", Request.Form("billTo_city"))
      oRequest.Add("billTo_state", Request.Form("billTo_state"))
      oRequest.Add("billTo_postalCode", Request.Form("billTo_postalCode"))
      oRequest.Add("billTo_country", Request.Form("billTo_country"))
      oRequest.Add("billTo_email", Request.Form("billTo_email"))
      oRequest.Add("card_accountNumber", Request.Form("card_accountNumber"))
      oRequest.Add("card_expirationMonth", Request.Form("card_expirationMonth"))
      oRequest.Add("card_expirationYear", Request.Form("card_expirationYear"))
      oRequest.Add("item_0_unitPrice", Request.Form("item_0_unitPrice"))
      oRequest.Add("purchaseTotals_currency", "USD")

      'Declare the reply hashtable
      Dim varReply As New Hashtable

      'Run the transaction
      varReply = CyberSource.Clients.NVPClient.RunTransaction(oRequest)

      'Print reply data to the browser
      Response.Write("reasonCode: " & varReply("reasonCode").ToString)
      Response.Write("<BR>Decision: " & varReply("decision").ToString)
      Response.Write("<BR>RequestID: " & varReply("requestID").ToString)
      Response.Write("<BR>Merchant Reference Code: " &
varReply("merchantReferenceCode").ToString)
   End Sub

End Class
```

CHAPTER 6

.NET 4.0 Client

Important

- The .NET 4.0 client for the Simple Order API is supported on 32-bit and 64-bit operating systems.
- If you are building an application to sell to others, see Appendix A, "Using the Client Application Fields," on page 390. This appendix has a list of API fields you can use in your request that describe the application, its version, and its user. If you use these fields in your request, you can view their values in the Transaction Search Details window of the Business Center.

Choosing an API and Client

API Variation

With this client package, you can use any of the three variations of the Simple Order API:

- Name-value pairs, which are simpler to use than XML
- XML, which requires you to create and parse XML documents
- SOAP (Simple Object Access Protocol) 1.1, which provides an object-oriented interface

The test that you run immediately after installing the client uses name-value pairs.

Client Versions

CyberSource updates the Simple Order API on a regular basis to introduce new API fields and functionality. To identify the latest version of the API, go to https://ics2wsa.ic3.com/commerce/1.x/transactionProcessor.

This represents the version of the server-side code for the CyberSource services.

If a new version of the API has been released, but CyberSource has not yet updated the .NET client to use this new version, you can manually update the client to use a different version. See "Updating the Client to Use a Later API Version," page 228.

Basic C# Program Example

The following example shows the primary code required to send a SOAP request for credit card authorization and process the reply. See "Using SOAP," page 251 for more information.

```
using CyberSource.Soap;
using CyberSource.Soap.CyberSourceWS;
using System;
using System.Configuration;
using System.Net;
using System.Web.Services.Protocols;
namespace Sample {
  class Sample {
    static void Main(string[] args) {
      RequestMessage request = new RequestMessage();
      request.merchantID = "infodev";

      // we want to do Credit Card Authorization in this sample
      request.ccAuthService = new CCAuthService();
      request.ccAuthService.run = "true";
      // add required fields
      request.merchantReferenceCode = "148705832705344";
      BillTo billTo = new BillTo();
      billTo.firstName = "Jane";
      billTo.lastName = "Smith";
      billTo.street1 = "1295 Charleston Road";
      billTo.city = "Mountain View";
      billTo.state = "CA";
      billTo.postalCode = "94043";
      billTo.country = "US";
      billTo.email = "jsmith@example.com";
      request.billTo = billTo;
      Card card = new Card();
      card.accountNumber = "4111111111111111";
      card.expirationMonth = "12";
      card.expirationYear = "2010";
      request.card = card;
      PurchaseTotals purchaseTotals = new PurchaseTotals();
      purchaseTotals.currency = "USD";
      request.purchaseTotals = purchaseTotals;

      // there is one item in this sample
      request.item = new Item[1];
      Item item = new Item();
      item.id = "0";
      item.unitPrice = "29.95";
      request.item[0] = item;
      // See "Interpreting the Reply," page 255 for details about

      // processing the reply for a SOAP transaction.
      try {
        ReplyMessage reply = Client.RunTransaction( request );
      } catch (CryptographicException ce) {
        Console.WriteLine( ce.ToString() );
      } catch (MessageSecurityException mse) {
        Console.WriteLine( mse.ToString() );
      } catch (WebException we) {
        Console.WriteLine( we.ToString() );
      } catch (Exception e) {
        Console.WriteLine( e.ToString() );
      }
    }
  }
}
```

Installing and Testing the Client

Minimum System Requirements

- Microsoft Windows 2000 or later
- .NET Framework 4.0 or later
- Microsoft Visual Studio 2010

Important

Failure to configure your client API host to a unique, public IP address will cause inconsistent transaction results.

The client API request ID algorithm uses a combination of IP address and system time, along with other values. In some architectures this combination might not yield unique identifiers.

Transaction Security Keys

The first thing you must do is create your security key. The client uses the security key to add a digital signature to every request that you send. This signature helps ensure that no one else can use your CyberSource account to process orders. You specify the location of your key when you configure the client.

Important

You must generate two transaction security keys—one for the CyberSource production environment and one for the test environment. For information about generating and using security keys, see *Creating and Using Security Keys* (PDF | HTML).

Warning

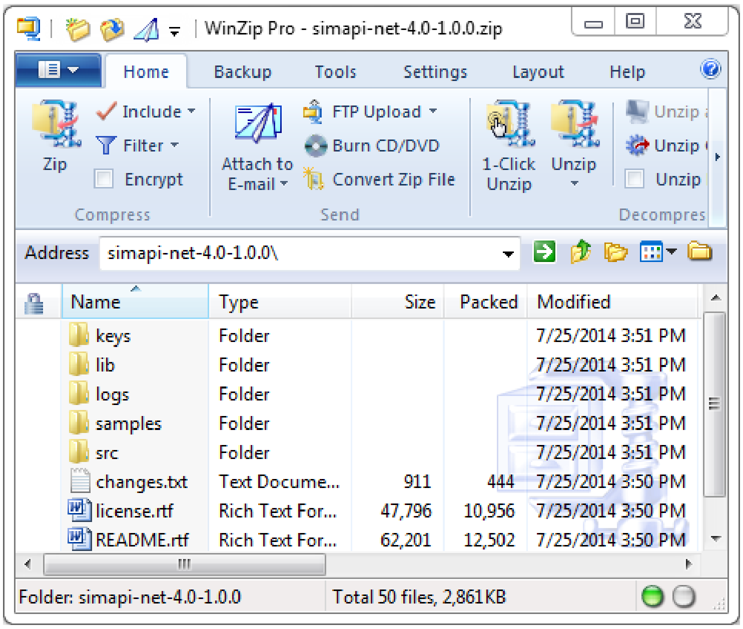
You must protect your security key to ensure that your CyberSource account is not compromised.

Installing for the First Time

To install the client for the first time:

Step 1 Go to the client downloads page on the Support Center and download the zip file that contains the latest version of the client.

Step 2 Unzip the downloaded file to the location of your choice:



Address simapi-net-4.0-1.0.0\

Name	Type	Size	Packed	Modified
keys	Folder			7/25/2014 3:51 PM
lib	Folder			7/25/2014 3:51 PM
logs	Folder			7/25/2014 3:51 PM
samples	Folder			7/25/2014 3:51 PM
src	Folder			7/25/2014 3:51 PM
changes.txt	Text Docume...	911	444	7/25/2014 3:50 PM
license.rtf	Rich Text For...	47,796	10,956	7/25/2014 3:50 PM
README.rtf	Rich Text For...	62,201	12,502	7/25/2014 3:50 PM

Folder: simapi-net-4.0-1.0.0 Total 50 files, 2,861KB

Step 3 Test the client. See "Using the Test Applications," page 223.

The client is installed and tested. You are ready to create your own code for requesting CyberSource services. Finish reading this section, and move on to one of these sections:

- "Using Name-Value Pairs," page 230 if you plan to use name-value pairs
- "Using XML," page 239 if you plan to use XML
- "Using SOAP," page 251 if you plan to use SOAP

Upgrading from a Previous Version

The .NET 4.0 Simple Order API client is a pure .NET client without dependencies outside of the .NET 4.0 Framework. It is simplified in comparison to previous Simple Order API .NET clients because it does not require the Microsoft Web Services Enhancements (WSE) and it does not use the CyberSource security libraries.

Previous versions of the Cybersource.Clients.dll required that you register CybsWSSecurity.dll as a COM object. The CybsWSSecurity.dll had dependencies on many other dynamic-link libraries (DLLs). Because the .NET 4.0 Simple Order API client does not use the CyberSource security libraries, you can remove or unregister the following DLLs:

- CybsWSSecurity.dll (unregister)
- CybsWSSecurityIOP.dll
- CyberSource.WSSecurity.dll
- domsupport_1_4_0.dll
- Msvcp60.dll
- platformsupport_1_4_0.dll
- spapache.dll
- xalandom_1_4_0.dll
- xalansourcetree_1_4_0.dll
- xerces-c_2_1_0.dll
- xercesparserliaison_1_4_0.dll
- xmlsupport_1_4_0.dll
- xpath_1_4_0.dll

Migrating from .NET Framework 1.x

To migrate from a .NET Framework 1.x client:

Step 1 Replace the old DLLs with the ones from this package.

Step 2 In your project, remove references to the previous CyberSource DLLs.

Step 3 Add a reference to `CyberSource.Clients.dll`.

Step 4 In your request code, make the following changes:

a Replace the referenced CyberSource namespaces with this one:

```
CyberSource.Clients
```

b If you use the SOAP client, add the following namespace:

```
CyberSource.Clients.SoapWebReference
```

Example In C#, with the SOAP client, you now have:

```
using CyberSource.Clients.
using CyberSource.Clients.SoapWebReference; /* for SOAP client
only */
```

Step 5 Follow the instructions for migrating from .NET Framework 2.X.

Migrating from .NET Framework 2.x

To migrate from a .NET Framework 2.x client:

Step 1 Follow the installation instructions in "Installing for the First Time," page 220.

Step 2 Open your project in Visual Studio 2010. If necessary, use the conversion wizard to update your project from Visual Studio 2005 to Visual Studio 2010.

Step 3 In your project properties, set the target framework to **.NET Framework 4**.

Step 4 Make sure that your reference to CyberSource.Clients points to the new .NET 4.0 version of the DLL. You must use the DLLs that you installed in Step 1.

Step 5 Remove references to System.Web.Services and remove the following namespace from your code:

```
using System.Web.Services.Protocols
```

Step 6 If your code contains catch statements that use `SignException`, change them to use `CryptographicException` instead. Making this change requires that you add a reference to System.Security and add the following namespace to your code:

```
using System.Security.Cryptography
```

Important

- For SOAP and name-value pair (NVP) clients only:

  Remove any catch statements that use `SoapHeaderException` or `SoapBodyException`.

- For SOAP clients only:

  Consider replacing these exceptions with appropriate Windows Communication Foundation (WCF) services exceptions such as `MessageSecurityException`, `EndpointNotFoundException`, or `ChannelTerminatedException` depending on your requirements. Then you must add a reference to `System.ServiceModel` and add the following namespaces to your code:

  ```
  using System.ServiceModel;
  using System.ServiceModel.Security;
  ```

You have successfully upgraded your client to the new version.

Using the Test Applications

Each type of client variation—name-value pair, XML, and SOAP—includes a pre-compiled test application. You can use these test applications to ensure that the client was installed correctly. The applications request both credit card authorization and capture.

The test applications and their source code are installed in the `samples` directory. The `bin` subdirectory contains the pre-compiled binaries. The `src` subdirectory contains the source code and Visual Studio project files.

Configuring the Test Applications

Before you run a test application, you must edit its application settings file. The following table describes all the configuration fields that you can use in this file.

Note

Configuration settings supported by the latest 1.x.x version are still supported. However, CyberSource recommends that you use the following new settings for this and future versions.

Table 35 Fields in the Settings File

Field Name	Description	Required/ Optional
cybs.connectionLimit	Maximum number of allowed concurrent connections between the client and CyberSource's server. For more information on this field and alternate ways to set the connection limits, see "Setting the Connection Limit," page 261.	Optional

Table 35 Fields in the Settings File (Continued)

Field Name	Description	Required/ Optional
cybs.keysDirectory	Directory that contains the pkcs12 security key file. For example: `c:\keys\`	Required
cybs.merchantID	Your CyberSource merchant ID. You can override this value by providing the merchantID field in the request itself. The merchant ID is case sensitive.	Optional
cybs. sendToProduction	Flag that indicates whether the transactions for this merchant should be sent to the production server. Use one of these values: ■ `false`: Do not send to the production server; send to the test server (default setting). ■ `true`: Send to the production server. **Note** Make sure that if your merchant ID is configured to use the test mode, you send requests to the test server.	Required
cybs.keyFilename	Name of the security key file name for the merchant in the format <***security_key_filename***>.p12.	Optional
cybs.serverURL	Alternate server URL to use. For more information, see "Configuring Your Settings for Multiple Merchants," page 226. Give the complete URL because it will be used exactly as you specify.	Optional
cybs.enableLog	Flag directing the client to log transactions and errors. Use one of these values: ■ `false`: Do not enable logging (default setting). ■ `true`: Enable logging. **Important** Logging can cause very large log files to accumulate. Therefore, CyberSource recommends that you use logging only when troubleshooting problems. To comply with all Payment Card Industry (PCI) and Payment Application (PA) Data Security Standards regarding the storage of credit card and card verification number data, the logs that are generated contain only masked credit card and card verification number data (CVV, CVC2, CVV2, CID, CVN). Follow these guidelines: ■ Use debugging temporarily for diagnostic purposes only. ■ If possible, use debugging only with test credit card numbers. ■ Never store clear text card verification numbers. ■ Delete the log files as soon as you no longer need them. ■ Never send email to CyberSource containing personal and account information, such as customers' names, addresses, card or check account numbers, and card verification numbers. For more information about PCI and PABP requirements, see www.visa.com/cisp.	Optional

Table 35 Fields in the Settings File (Continued)

Field Name	Description	Required/ Optional
cybs.logDirectory	Directory to which to write the log file. Note that the client will not create this directory for you; you must specify an existing directory. The client includes a `logs` directory that you can use. Include the path. For example: `c:\simapi-net-2.0.0\logs`.	Required if **cybs. enableLog** is true
cybs.logFilename	Name of the log file. The client uses `cybs.log` by default.	Optional
cybs.logMaximumSize	Maximum size in megabytes for the log file. The default value is `10`. When the log file reaches the specified size, it is archived into `cybs.log.`***<yyyymmdd*****`T`*****hhmmssxxx>*** and a new log file is started. The ***xxx*** indicates milliseconds.	Optional
cybs.timeout	Length of time-out in seconds. The default is 130.	Optional
cybs.proxyURL	URL of a proxy server. For example: `https://proxy.example.com:4909`	Optional
cybs.proxyUser	User name for the proxy server.	Optional
cybs.proxyPassword	Password for the proxy server.	Optional

To test applications:

Step 1 Decide which test application you want to run, such as `SoapSample.exe`.

Step 2 Using a text editor, open the settings file for the test application.

The settings file has the same name as the test application, with the extension `config` appended to the name. For example, `SoapSample.exe.config`.

Step 3 Find the `cybs.merchantID` field and change its value to your CyberSource merchant ID.

For example, if your merchant ID is `widgetsinc`, change the field to `<add key="cybs.merchantID" value="`**`widgetsinc`**`"/>`.

The merchant ID is case sensitive.

Step 4 Find the `cybs.keysDirectory` field and change its value to the directory that contains your security key.

For example, if your key is in `c:\keys\`, change the field to `<add key="cybs.keysDirectory" value="`**`c:\keys\`**`"/>`.

Step 5 Edit other fields as necessary.

See Table 35, "Fields in the Settings File," on page 223 for a complete list.

Step 6 Save and close the settings file.

Configuring Your Settings for Multiple Merchants

If you have multiple merchant IDs, or if you are a reseller handling multiple merchants, you can configure the settings to allow different configurations for different merchant IDs.

To specify the settings for a specific merchant, prefix all settings, except for `cybs.merchantID` and the `cybs.proxy*`, with `<merchantID>.` The `cybs.proxy*` wildcard refers to the `proxyURL`, `proxyUser`, `proxyPassword` settings.

Example **You have a new merchant with merchant ID of `NewMerchant`. To send only test transactions for this merchant, you can set all requests for `NewMerchant` to go to the test server:**

```
<add key="cybs.NewMerchant.sendToProduction" value="false"/>
<add key="cybs.sendToProduction" value="true"/>
```

With the second line of the example, the client will send all other requests to the production server.

Running the Test Applications

To run test applications:

Step 1 Open a Windows command-line shell.

Step 2 Change to the directory where the test application is located.

Step 3 Type the name of the test application, then press Enter.

The test application requests an CyberSource service, interprets the reply, and prints information about the result. If you receive a .NET exception, use the error message to debug the problem.

Deploying the Client to Another Computer

To deploy the client to another computer without running the installer provided by CyberSource, you must include all the files from the `lib` directory in your custom installer and then run it. Then the client is ready to be used on the computer.

Going Live

When you complete all of your system testing and are ready to accept real transactions from consumers, your deployment is ready to *go live*.

Note After your deployment goes live, use real card numbers and other data to test every card type you support. Because these are real transactions in which you are buying from yourself, use small monetary amounts to do the tests. Process an authorization, then capture the authorization, and later refund the money. Use your bank statements to verify that money is deposited into and withdrawn from your merchant bank account as expected. If you have more than one CyberSource merchant ID, test each one separately.

CyberSource Essentials Merchants

If you use CyberSource Essentials services, you can use the Business Center site to go live. For a description of the process of going live, see the “Steps for Getting Started” section in *Getting Started with CyberSource Essentials*.

Important You must also configure your client so that it sends transactions to the production server and not the test server. See the description of the configuration setting "cybs. sendToProduction," page 224.

CyberSource Advanced Merchants

If you use CyberSource Advanced services, see the “Steps for Getting Started” chapter in *Getting Started with CyberSource Advanced* for information about going live.

When your deployment goes live, your CyberSource account is updated so that you can send transactions to the CyberSource production server. If you have not already done so, you must provide your banking information to CyberSource so that your processor can deposit funds to your merchant bank account.

After CyberSource confirms that your account is live, make sure that you update your system so that it can send requests to the production server (`ics2wsa.ic3.com`) using your security key for the production environment. The test server (`ics2wstesta.ic3.com`) cannot be used for real transactions. For more information about sending transactions to the production server, see the description of the configuration setting "cybs. sendToProduction," page 224.

Updating the Client to Use a Later API Version

CyberSource periodically updates the Simple Order API. You can update your existing client to work with the new API version. For a list of the available API versions, go to https://ics2wsa.ic3.com/commerce/1.x/transactionProcessor.

Alternately, if a new client is available that works with the later API version, you can download that new client.

Note The new client may have new functionality unrelated to the changes in the API. Read the release notes in the `CHANGES` file to determine if the new client contains new functionality that you want to use.

Name-Value Pair Client

To update a name-value pair client:

Step 1 Load `src\CyberSource.Clients.sln` in Visual Studio 2010.

Step 2 In the Solution Explorer, locate the Service References folder.

Step 3 Right-click **NVPWebReference** and choose **Configure Service Reference**.

Step 4 Update the Address field with the New WSDL URL. Typically, only the version number at the end of the URL needs to be updated.

Step 5 Build the Release configuration.

Step 6 Save a copy of the original `CyberSource.Clients.dll` and then replace it with the newly built `CyberSource.Clients.dll`.

SOAP Client

To update a SOAP client:

Step 1 Load `src\CyberSource.Clients.sln` in Visual Studio 2010.

Step 2 In the Solution Explorer, locate the Service References folder.

Step 3 Right-click **SoapWebReference** and choose **Configure Service Reference**.

Step 4 Update the Address field with the New WSDL URL. Typically, only the version number at the end of the URL needs to be updated.

Step 5 Build the Release configuration.

Step 6 Save a copy of the original `CyberSource.Clients.dll` and then replace it with the newly built `CyberSource.Clients.dll`.

XML Client

Updating the client is unnecessary. Start using the new namespace URI in your input XML documents. The client automatically uses the specified version.

Using Name-Value Pairs

This section explains how to request CyberSource services by using name-value pairs.

Requesting CyberSource Services

To request CyberSource services, write code that:

- Collects information for the CyberSource services that you will use
- Assembles the order information into requests
- Sends the requests to the CyberSource server
- Processes the reply information

Important The CyberSource servers do not support persistent HTTP connections.

The instructions in this section explain how to write C# programs that request CyberSource services. For a list of API fields to use in your requests, see "Related Documents," page 21.

Creating and Sending the Request

To use any CyberSource service, you must create and send a request that includes the required information for that service.

The example developed in the following sections shows basic code for requesting CyberSource services. In this example, Jane Smith is buying an item for $29.95.

Important The code in this section’s examples is incomplete. For complete sample programs, see the source code in the client’s samples\src\nvp directory.

Creating a New Visual Studio .NET Project

To get started, create a new project in Visual Studio .NET, and add a reference to the client library, `CyberSource.Clients.dll`, which is located in the client’s lib directory.

Importing the Client Classes

In the code for your application, add the following import statements:

```
using CyberSource.Clients;
using System;
using System.Collections;
using System.Net;
using System.Security.Cryptography;
using System.ServiceModel;
using System.ServiceModel.Security;
```

Creating an Empty Request

You next create a hashtable that holds the request fields:

```
Hashtable request = new Hashtable();
```

Adding the Merchant ID

You next optionally add your CyberSource merchant ID to the request:

```
request.Add( "merchantID", "infodev" );
```

This value overrides any value you set with the merchantID configuration setting (see Table 35, "Fields in the Settings File," on page 223). The merchant ID is case sensitive.

Adding Services to the Request

You next indicate the service that you want to use by adding a field to the request. For example, to request a credit card authorization:

```
request.Add( "ccAuthService_run", "true" );
```

Requesting a Sale

You can request multiple services by adding additional fields to the request. For example, if you fulfill the order immediately, you can request a credit card authorization and capture together (also referred to as a “sale”):

```
request.Add( "ccAuthService_run", "true" );
request.Add( "ccCaptureService_run", "true" );
```

Adding Service-Specific Fields to the Request

You next add the fields that are used by the services you are requesting. If you request multiple services and they share common fields, you must add the field once only.

```
request.Add( "billTo_firstName", "Jane" );
request.Add( "billTo_lastName", "Smith" );
request.Add( "card_accountNumber", "4111111111111111" );
request.Add( "item_0_unitPrice", "29.95" );
```

The previous example shows only a partial list of the fields you must send. Refer to "Requesting CyberSource Services," page 230 for information about the guides that list all of the fields for the services that you are requesting.

Sending the Request

You next send the request to CyberSource, store the reply in a new hash table, and catch several exceptions that you might receive:

```
try {
  Hashtable reply = NVPClient.RunTransaction( request );
  SaveOrderState();
  // "Using the Decision and Reason Code," page 234 describes the ProcessReply
  // method.
  ProcessReply( reply );
} catch (CryptographicException ce) {
  SaveOrderState();
  Console.WriteLine( ce.ToString() );
} catch (WebException we) {
  SaveOrderState();
  /*
   * Some types of WebException indicate that the transaction may have been
   * completed by CyberSource. The sample code shows how to identify these
   * exceptions. If you receive such an exception, and your request included a
   * payment service, you should use the CyberSource transaction search screens to
   * determine whether the transaction was processed.
   */
  Console.WriteLine( we.ToString() );
}private static void SaveOrderState() {
  /*

   * This is where you store the order state in your system for post-transaction
   * analysis. Be sure to store the consumer information, the values of the reply
   * fields, and the details of any exceptions that occurred.
   */
}
```

In the preceding example, when an exception occurs, the exception is printed to the console. Your web store should also display a message to the consumer indicating that you were unable to process the order. The sample code for the name-value pair client shows you how to provide feedback to the consumer.

Also, if the transaction fails, and the request did not include any payment services, you may be able to resend the transaction. The sample code for the name-value pair client shows you how to do this.

Interpreting the Reply

After the CyberSource server processes your request, it sends a reply message that contains information about the services you requested. You receive different fields depending on the services you request and the outcome of each service.

To use the reply information, you must integrate it into your system and any other system that uses that data. For example, you can store the reply information in a database and send it to other back office applications.

You must write an error handler to process the reply information that you receive from CyberSource. Do not show the reply information directly to consumers. Instead, present an appropriate response that tells consumers the result.

Important Because CyberSource may add reply fields and reason codes at any time, you should parse the reply data according to the names of the fields instead of their order in the reply.

The most important reply fields to evaluate are the following:

- **decision**: A one-word description of the results of your request. The decision is one of the following:
  - `ACCEPT` if the request succeeded
  - `REJECT` if one or more of the services in the request was declined
  - `REVIEW` if you use CyberSource Decision Manager and it flags the order for review. See "For CyberSource Advanced Merchants: Handling Decision Manager Reviews," page 236 for more information.
  - `ERROR` if there was a system error. See "Retrying When System Errors Occur," page 238 for important information about handling retries in the case of system errors.
- **reasonCode**: A numeric code that provides more specific information about the results of your request.

You also receive a reason code for each service in your request. You can use these reason codes to determine whether a specific service succeeded or failed. If a service fails, other services in your request may not run. For example, if you request a credit card

authorization and capture, and the authorization fails, the capture does not run. The reason codes for each service are described in the *Credit Card Services User Guide* for CyberSource Essentials merchants or in the service developer guide for CyberSource Advanced merchants.

Important CyberSource reserves the right to add new reason codes at any time. If your error handler receives a reason code that it does not recognize, it should use the decision to interpret the reply.

Using the Decision and Reason Code

The following example shows how you can use the decision and the reason code to display an appropriate message to the consumer.

```
private static bool ProcessReply( Hashtable reply ) {
  string template = GetTemplate(
    ((string)reply["decision"]).ToUpper() );
  string content  = GetContent( reply );

  // This example writes the message to the console. Choose an appropriate display
  // method for your own application.
  Console.WriteLine( template, content );
}

private static string GetTemplate( string decision ) {
  // Retrieves the text that corresponds to the decision.

  if ("ACCEPT".Equals( decision )) {
    return( "The order succeeded.{0}" );
  }

  if ("REJECT".Equals( decision )) {
    return( "Your order was not approved.{0}" );
  }

  // ERROR, or an unknown decision
  return( "Your order could not be completed at this time.{0}" +
          "\nPlease try again later." );
}
private static string GetContent( Hashtable reply ) {
  /*
   * Uses the reason code to retrieve more details to add to the template.
   *
   * The messages returned in this example are meant to demonstrate how to
   * retrieve the reply fields. Your application should display user-friendly
   * messages.
   */
  int reasonCode = int.Parse( (string) reply["reasonCode"] );
  switch (reasonCode) {
    // Success
    case 100:
      return( "\nRequest ID: " + reply["requestID"] );

    // Missing field or fields

    case 101:
      return( "\nThe following required fields are missing: " +
              EnumerateValues( reply, "missingField" ) );

    // Invalid field or fields
    case 102:
      return( "\nThe following fields are invalid: " +
              EnumerateValues( reply, "invalidField" ) );

    // Insufficient funds
    case 204:
      return( "\nInsufficient funds in the account. Please use a " +
              "different card or select another form of payment." );

    // Add additional reason codes here that you must handle more specifically.

    default:
      // For all other reason codes, such as unrecognized reason codes, or codes
      // that do not require special handling, return an empty string.
      return( String.Empty );
  }
}
private static string EnumerateValues( Hashtable reply,
                                       string fieldName ) {
  System.Text.StringBuilder sb = new System.Text.StringBuilder();
  string val = "";
  for (int i = 0; val != null; ++i) {
    val = (string) reply[fieldName + "_" + i];
    if (val != null) {
      sb.Append( val + "\n" );
    }
  }

  return( sb.ToString() );
}
```

For CyberSource Advanced Merchants: Handling Decision Manager Reviews

The information in this section applies only to CyberSource Advanced merchants.

If you use CyberSource Decision Manager, you may also receive the `REVIEW` value in the **decision** field. `REVIEW` means that Decision Manager has marked the order for review based on how you configured the Decision Manager rules.

If you will be using Decision Manager, you have to determine how to handle the new `REVIEW` value. Ideally, you will update your order management system to recognize the `REVIEW` response and handle it according to your business rules. If you cannot update your system to handle the `REVIEW` response, CyberSource recommends that you choose one of these options:

- If you authorize and capture the credit card payment at the same time, treat the `REVIEW` response like a `REJECT` response. Rejecting any orders that are marked for review may be appropriate if your product is a software download or access to a Web site. If supported by your processor, you may also want to reverse the authorization.

- If you approve the order after reviewing it, convert the order status to `ACCEPT` in your order management system. You can request the credit card capture without requesting a new authorization.

- If you approve the order after reviewing it but cannot convert the order status to `ACCEPT` in your system, request a new authorization for the order. When processing this new authorization, you must disable Decision Manager. Otherwise the order will be marked for review again. For details about the API field that disables Decision Manager, see the *Decision Manager Developer Guide Using the Simple Order API* (PDF | HTML) or the *Decision Manager Developer Guide Using the SCMP Order API* (PDF | HTML).

Alternately, you can specify a custom business rule in Decision Manager so that authorizations originating from a particular internal IP address at your company are automatically accepted.

If supported by your processor, you may want to reverse the original authorization.

Requesting Multiple Services

When you request multiple services in one request, CyberSource processes the services in a specific order. If a service fails, CyberSource does not process the subsequent services in the request.

For example, in the case of a sale (a credit card authorization and a capture requested together), if the authorization service fails, CyberSource will not process the capture service. The reply you receive only includes reply fields for the authorization.

Many CyberSource services include “ignore” fields that tell CyberSource to ignore the result from the first service when deciding whether to run the subsequent services. In the case of the sale, even though the issuing bank gives you an authorization code, CyberSource might decline the authorization based on the AVS or card verification results. Depending on your business needs, you might choose to capture these types of declined authorizations anyway. You can set the **businessRules_ignoreAVSResult** field to `"true"` in your combined authorization and capture request:

```
request.put( "businessRules_ignoreAVSResult", "true" );
```

This tells CyberSource to continue processing the capture even if the AVS result causes CyberSource to decline the authorization. In this case you would then get reply fields for both the authorization and the capture in your reply.

Note

You are charged only for the services that CyberSource performs.

Retrying When System Errors Occur

You must design your transaction management system to include a way to correctly handle CyberSource system errors. Depending on which payment processor is handling the transaction, the error may indicate a valid CyberSource system error, or it may indicate a processor rejection because of some type of invalid data. In either case, CyberSource recommends that you do not design your system to retry sending a transaction many times in the case of a system error.

Instead, CyberSource recommends that you retry sending the request only two or three times with successively longer periods of time between each retry. For example, after the first system error response, wait 30 seconds and then retry sending the request. If you receive the same error a second time, wait one minute before you send the request again. Depending on the situation, you may decide you can retry sending the request after a longer time period. Determine what is most appropriate for your business situation.

If after several retry attempts you are still receiving a system error, it is possible that the error is actually being caused by a processor rejection and not a CyberSource system error. In that case, CyberSource suggest that you either:

- Search for the transaction in the Business Center (depending on which one you normally use), look at the description of the error on the Transaction Detail page, and call your processor to determine if and why they are rejecting the transaction.

- Contact CyberSource Customer Support to confirm whether your error is truly caused by a CyberSource system issue.

If TSYS Acquiring Solutions is your processor, you may want to follow the first suggestion as there are several common TSYS Acquiring Solutions processor responses that are returned to you as system errors and that only TSYS Acquiring Solutions can address.

Creating an Application Settings File

After you finish writing code for your integration, you must create an application settings file. This file must contain at least the following information:

- The directory that contains your security key
- The location of the CyberSource server

See Table 35, "Fields in the Settings File," on page 223 for a complete list of settings.

You can use the settings files that come with the sample applications as a starting point for your own settings file. See "Configuring the Test Applications," page 223 for more information.

Using XML

This section explains how to request CyberSource services by using XML.

Requesting CyberSource Services

To request CyberSource services, write code that:

- Collects information for the CyberSource services that you will use
- Assembles the order information into requests
- Sends the requests to the CyberSource server
- Processes the reply information

Important The CyberSource servers do not support persistent HTTP connections.

The instructions in this section explain how to write C# programs that request CyberSource services. For a list of API fields to use in your requests, see "Related Documents," page 21.

Creating a Request Document

The XML client allows you to create an XML request document using any application, then send the request to CyberSource. For example, if you have a customer relationship management (CRM) system that uses XML to communicate with other systems, you can use the CRM system to generate request documents.

The request document must validate against the XML schema for CyberSource transactions. To view the schema, go to

https://ics2wsa.ic3.com/commerce/1.x/transactionProcessor

and look at the `xsd` file for the version of the Simple Order API you are using.

Important

Make sure that the elements in your document appear in the correct order. If they do not, your document will not validate, and your request will fail.

The following example shows a basic XML document for requesting CyberSource services. In this example, Jane Smith is buying an item for $29.95.

Note

The XML document in this example is incomplete. For complete examples, see `sample.xml` in the client's `samples\bin` directory.

Creating an Empty Request

Add the XML declaration and the document's root element:

```
<?xml version="1.0" encoding="utf-8"?>
<requestMessage xmlns="urn:schemas-cybersource-com:transaction-data-
1.17">
</requestMessage>
```

Make sure that the API version specified at the end of the namespace is correct. For example, to communicate with version 1.19, you must use the namespace `urn:schemas-cybersource-com:transaction-data-1.19`. When you must update the API version, see "Updating the Client to Use a Later API Version," page 228.

Note

The XML document that you receive in the reply always has the prefix `c:`, for example: `xmlns:c="urn:schemas-cybersource-com:transaction-data-1.17"`. Make sure you use an XML parser that supports namespaces.

Adding the Merchant ID

Optionally, you can add the CyberSource merchant ID to the request:

```
<?xml version="1.0" encoding="utf-8"?>
<requestMessage xmlns="urn:schemas-cybersource-com:transaction-data-
1.17">
  <merchantID>infodev</merchantID>
</requestMessage>
```

This value overrides any value that you set with the merchantID configuration setting. For more information about the merchantID configuration setting, see Table 35, "Fields in the Settings File," on page 223. The merchant ID is case sensitive.

Adding Services to the Request

You next indicate the service that you want to use by creating an element for that service in the request, then setting the element’s `run` attribute to `true`. For example, to request a credit card authorization:

```
<?xml version="1.0" encoding="utf-8"?>
<requestMessage xmlns="urn:schemas-cybersource-com:transaction-data-
1.15">
  <merchantID>infodev</merchantID>
  <ccAuthService run="true"/>
</requestMessage>
```

Requesting a Sale

You can request multiple services by creating additional elements. For example, if you fulfill the order immediately, you can request a credit card authorization and capture together (referred to as a “sale”):

```
<?xml version="1.0" encoding="utf-8"?>
<requestMessage xmlns="urn:schemas-cybersource-com:transaction-data-
1.17">
  <merchantID>infodev</merchantID>
  <ccAuthService run="true"/>
  <ccCaptureService run="true"/>
</requestMessage>
```

Adding Service-Specific Fields to the Request

You next add the fields that are used by the services you are requesting. Most fields are child elements of container elements; for example, a `<card>` element contains the consumer's credit card information.

```
<?xml version="1.0" encoding="utf-8"?>
<requestMessage xmlns="urn:schemas-cybersource-com:transaction-data-
1.15">
  <merchantID>infodev</merchantID>
  <billTo>
    <firstName>Jane</firstName>
    <lastName>Smith</lastName>
  </billTo>
  <item id="0">
  </card>
    <unitPrice>29.95</unitPrice>
  </item>
  <card>
    <accountNumber>4111111111111111</accountNumber>
  <ccAuthService run="true"/>
</requestMessage>
```

The example above shows only a partial list of the fields you must send. Refer to "Related Documents," page 21 for information about the guides that list all of the fields for the services that you are requesting.

Sending the Request

Once you have created an XML request document, you can use a .NET application to send the request to CyberSource. The example that follows is written in C#.

Note

The code in this section's examples is incomplete. For complete sample programs, see the source code in the client's `samples\src\xml` directory.

Creating a New Visual Studio .NET Project

To start, create a new project in Visual Studio .NET. Then you must add a reference to the client library, `CyberSource.Clients.dll` (located in the client's lib directory) and to the .NET Framework `System.Security.dll` library.

Importing the Client Classes

In the code for your application, add the following import statements:

```
using CyberSource.Clients;
using System;
using System.Net;
using System.Xml;
using System.Security.Cryptography
```

Sending the Request

You next read the XML request document, send the request to CyberSource, store the reply in a new `XmlDocument` object, and catch several exceptions that you might receive:

```
try {
  XmlDocument request = new XmlDocument();
  request.Load( "MyXmlDocument.xml" );

  XmlDocument reply = XmlClient.RunTransaction( request );
  SaveOrderState();
  // "Using the Decision and Reason Code," page 234 describes the ProcessReply
  // method.
  ProcessReply( reply );
} catch (CryptographicException ce) {
  SaveOrderState();
  Console.WriteLine( ce.ToString() );
} catch (FaultException fe) {
  SaveOrderState();
  /*
   * Some types of FaultException indicate that the transaction may have been
   * completed by CyberSource. The sample code shows how to identify these
   * exceptions. If you receive such an exception, and your request included a
   * payment service, you should use the CyberSource transaction search screens to
   * determine whether the transaction was processed.
   */
  Console.WriteLine( fe.ToString() );
} catch (WebException we) {
  SaveOrderState();
  /*
   * Some types of WebException indicate that the transaction may have been completed
   * by CyberSource. The sample code shows how to identify these exceptions. If you
   * receive such an exception, and your request included a payment service, you
   * should use the CyberSource transaction search screens to determine whether the
   * transaction was processed.
   */
  Console.WriteLine( we.ToString() );
}
```

```
private static void SaveOrderState() {
  /*
   * This is where you store the order state in your system for post-transaction
   * analysis. Be sure to store the consumer information, the values of the reply
   * fields, and the details of any exceptions that occurred.
   */
}
```

In the preceding example, when an exception occurs, the exception is printed to the console. Your web store should also display a message to the consumer indicating that you were unable to process the order. The sample code for the XML client shows you how to provide feedback to the consumer.

Also, if the transaction fails, and the request did not include any payment services, you may be able to resend the transaction. The sample code for the XML client shows you how to do this.

Interpreting the Reply

After the CyberSource server processes your request, it sends a reply message that contains information about the services you requested. You receive different fields depending on the services you request and the outcome of each service.

To use the reply information, you must integrate it into your system and any other system that uses that data. For example, you can store the reply information in a database and send it to other back office applications.

You must write an error handler to process the reply information that you receive from CyberSource. Do not show the reply information directly to consumers. Instead, present an appropriate response that tells consumers the result.

Important Because CyberSource may add reply fields and reason codes at any time, you should parse the reply data according to the names of the fields instead of their order in the reply.

The most important reply fields to evaluate are the following:

- **decision**: A one-word description of the results of your request. The decision is one of the following:
  - `ACCEPT` if the request succeeded
  - `REJECT` if one or more of the services in the request was declined
  - `REVIEW` if you use CyberSource Decision Manager and it flags the order for review. See "For CyberSource Advanced Merchants: Handling Decision Manager Reviews," page 248 for more information.
  - `ERROR` if there was a system error. See "Retrying When System Errors Occur," page 250 for important information about handling retries in the case of system errors.
- **reasonCode**: A numeric code that provides more specific information about the results of your request.

You also receive a reason code for each service in your request. You can use these reason codes to determine whether a specific service succeeded or failed. If a service fails, other services in your request may not run. For example, if you request a credit card authorization and capture, and the authorization fails, the capture does not run. The reason codes for each service are described in the *Credit Card Services User Guide* for CyberSource Essentials merchants or in the service developer guide for CyberSource Advanced merchants.

Important

CyberSource reserves the right to add new reason codes at any time. If your error handler receives a reason code that it does not recognize, it should use the decision to interpret the reply.

Using the Decision and Reason Code

The following example shows how you can use the decision and the reason code to display an appropriate message to the consumer.

```
private static bool ProcessReply( XmlDocument reply ) {
  // The following code allows you to use XPath with the CyberSource schema, which
  // uses a non-empty default namespace.
  XmlNamespaceManager nsmgr
    = new XmlNamespaceManager( reply.NameTable );
  nsmgr.AddNamespace( "cybs", Client.CYBS_NAMESPACE );

  XmlNode replyMessage
    = reply.SelectSingleNode( "cybs:replyMessage", nsmgr);

  string decision = replyMessage.SelectSingleNode(
    "cybs:decision/text()", nsmgr ).Value;

  string template = GetTemplate( decision.ToUpper() );
  string content  = GetContent( replyMessage, nsmgr );

  // This example writes the message to the console. Choose an appropriate display
  // method for your own application.
  Console.WriteLine( template, content );
}
private static string GetTemplate( string decision ) {
  // Retrieves the text that corresponds to the decision.
  if ("ACCEPT".Equals( decision )) {
    return( "The order succeeded.{0}" );
  }

}
  if ("REJECT".Equals( decision )) {
    return( "Your order was not approved.{0}" );
  }

  // ERROR, or an unknown decision
  return( "Your order could not be completed at this time.{0}" +
          "\nPlease try again later." );
private static string GetContent(
  XmlNode replyMessage, XmlNamespaceManager nsmgr ) {
  /*
   * Uses the reason code to retrieve more details to add to the template.
   *
   * The messages returned in this example are meant to demonstrate how to retrieve
   * the reply fields. Your application should display user-friendly messages.
   */
  string textVal = replyMessage.SelectSingleNode(
    "cybs:reasonCode/text()", nsmgr ).Value;
  int reasonCode = int.Parse( textVal );
  switch (reasonCode) {
    // Success
    case 100:
      return( "\nRequest ID: " +
        replyMessage.SelectSingleNode(
          "cybs:requestID/text()", nsmgr ).Value );
    // Missing field or fields
    case 101:
      return( "\nThe following required fields are missing: " +
              EnumerateValues( replyMessage.SelectNodes(
                "cybs:missingField/text()", nsmgr ) ) );

    // Invalid field or fields
    case 102:
      return( "\nThe following fields are invalid: " +
              EnumerateValues( replyMessage.SelectNodes(
                "cybs:invalidField/text()", nsmgr ) ) );
    // Insufficient funds
    case 204:
      return( "\nInsufficient funds in the account. Please use a " +
              "different card or select another form of payment." );

    // Add additional reason codes here that you must handle more specifically.

    default:
      // For all other reason codes (for example, unrecognized reason codes, or
      // codes that do not require special handling), return an empty string.
      return( String.Empty );
  }
}
private static string EnumerateValues( XmlNodeList nodes ) {
  System.Text.StringBuilder sb = new System.Text.StringBuilder();
  foreach (XmlNode node in nodes) {
    sb.Append( val + "\n" );
  }
  return( sb.ToString() );
}
```

For CyberSource Advanced Merchants: Handling Decision Manager Reviews

The information in this section applies only to CyberSource Advanced merchants.

If you use CyberSource Decision Manager, you may also receive the `REVIEW` value in the **decision** field. `REVIEW` means that Decision Manager has marked the order for review based on how you configured the Decision Manager rules.

If you will be using Decision Manager, you have to determine how to handle the new `REVIEW` value. Ideally, you will update your order management system to recognize the `REVIEW` response and handle it according to your business rules. If you cannot update your system to handle the `REVIEW` response, CyberSource recommends that you choose one of these options:

- If you authorize and capture the credit card payment at the same time, treat the `REVIEW` response like a `REJECT` response. Rejecting any orders that are marked for review may be appropriate if your product is a software download or access to a Web site. If supported by your processor, you may also want to reverse the authorization.

- If you approve the order after reviewing it, convert the order status to `ACCEPT` in your order management system. You can request the credit card capture without requesting a new authorization.

- If you approve the order after reviewing it but cannot convert the order status to `ACCEPT` in your system, request a new authorization for the order. When processing this new authorization, you must disable Decision Manager. Otherwise the order will be marked for review again. For details about the API field that disables Decision Manager, see the *Decision Manager Developer Guide Using the Simple Order API* (PDF | HTML) or the *Decision Manager Developer Guide Using the SCMP Order API* (PDF | HTML).

Alternately, you can specify a custom business rule in Decision Manager so that authorizations originating from a particular internal IP address at your company are automatically accepted.

If supported by your processor, you may want to reverse the original authorization.

Requesting Multiple Services

When you request multiple services in one request, CyberSource processes the services in a specific order. If a service fails, CyberSource does not process the subsequent services in the request.

For example, in the case of a sale (a credit card authorization and a capture requested together), if the authorization service fails, CyberSource will not process the capture service. The reply you receive only includes reply fields for the authorization.

Many CyberSource services include “ignore” fields that tell CyberSource to ignore the result from the first service when deciding whether to run the subsequent services. In the case of the sale, even though the issuing bank gives you an authorization code, CyberSource might decline the authorization based on the AVS or card verification results. Depending on your business needs, you might choose to capture these types of declined authorizations anyway. You can set the **businessRules_ignoreAVSResult** field to `"true"` in your combined authorization and capture request:

```
<businessRules>
    <ignoreAVSResult>true</ignoreAVSResult>
</businessRules>
```

This tells CyberSource to continue processing the capture even if the AVS result causes CyberSource to decline the authorization. In this case you would then get reply fields for both the authorization and the capture in your reply.

Note You are charged only for the services that CyberSource performs.

Retrying When System Errors Occur

You must design your transaction management system to include a way to correctly handle CyberSource system errors. Depending on which payment processor is handling the transaction, the error may indicate a valid CyberSource system error, or it may indicate a processor rejection because of some type of invalid data. In either case, CyberSource recommends that you do not design your system to retry sending a transaction many times in the case of a system error.

Instead, CyberSource recommends that you retry sending the request only two or three times with successively longer periods of time between each retry. For example, after the first system error response, wait 30 seconds and then retry sending the request. If you receive the same error a second time, wait one minute before you send the request again. Depending on the situation, you may decide you can retry sending the request after a longer time period. Determine what is most appropriate for your business situation.

If after several retry attempts you are still receiving a system error, it is possible that the error is actually being caused by a processor rejection and not a CyberSource system error. In that case, we suggest that you either:

- Search for the transaction in the Business Center (depending on which one you normally use), look at the description of the error on the Transaction Detail page, and call your processor to determine if and why they are rejecting the transaction.
- Contact CyberSource Customer Support to confirm whether your error is truly caused by a CyberSource system issue.

If TSYS Acquiring Solutions is your processor, you may want to follow the first suggestion as there are several common TSYS Acquiring Solutions processor responses that are returned to you as system errors and that only TSYS Acquiring Solutions can address.

Creating an Application Settings File

After you finish writing code for your integration, you must create an application settings file. This file must contain, at a minimum, the following information:

- The directory that contains your security key
- The location of the CyberSource server

See Table 35, "Fields in the Settings File," on page 223 for a complete list of settings.

You can use the settings files that come with the sample applications as a starting point for your own settings file. See "Configuring the Test Applications," page 223 for more information.

Using SOAP

This section explains how to request CyberSource services by using the Simple Object Access Protocol (SOAP).

Requesting CyberSource Services

To request CyberSource services, write code that:

- Collects information for the CyberSource services that you will use
- Assembles the order information into requests
- Sends the requests to the CyberSource server

Important

The CyberSource servers do not support persistent HTTP connections.

- Processes the reply information

The instructions in this section explain how to write C# programs that request CyberSource services. For a list of API fields to use in your requests, see "Related Documents," page 21.

Creating and Sending the Request

To use any CyberSource service, you must create and send a request that includes the required information for that service.

The following example shows basic code for requesting CyberSource services. In this example, Jane Smith is buying an item for $29.95.

Important

The code in this section's examples is incomplete. For complete sample programs, see the source code in the client's `samples\src\soap` directory.

Creating a New Visual Studio .NET Project

To get started, create a new project in Visual Studio .NET. Then you must add a reference to the client library, `CyberSource.Clients.dll` (located in the client's lib directory). You must also add references to the .NET Framework libraries `System.ServiceModel.dll` and `System.Security.dll`.

Importing the Client Classes

In the code for your application, add the following import statements:

```
using System;
using System.Net;
using System.Security.Cryptography;
using System.ServiceModel;
using System.ServiceModel.Security;
using CyberSource.Clients;
using CyberSource.Clients.SoapWebReference;
```

Creating an Empty Request

You next create a RequestMessage object that holds the request fields:

```
RequestMessage request = new RequestMessage();
```

Adding the Merchant ID

You next optionally add your CyberSource merchant ID to the request:

```
request.merchantID = "infodev";
```

This value overrides any value you set with the merchantID configuration setting (see Table 35, "Fields in the Settings File," on page 223). The merchant ID is case sensitive.

Adding Services to the Request

You next indicate the service that you want to use by creating an object for that service in the request, then setting the object's `run` property to `true`. For example, to request a credit card authorization:

```
request.ccAuthService = new CCAuthService();
request.ccAuthService.run = "true";
```

Requesting a Sale

You can request multiple services by creating additional objects. For example, if you fulfill the order immediately, you can request a credit card authorization and capture together (referred to as a “sale”):

```
request.ccAuthService = new CCAuthService();
request.ccAuthService.run = "true";
request.ccCaptureService = new CCCaptureService();
request.ccCaptureService.run = "true";
```

Adding Service-Specific Fields to the Request

You next add the fields that are used by the services you are requesting. Most fields are properties of additional objects; for example, a `Card` object contains the consumer's credit card information.

```
BillTo billTo = new BillTo();
billTo.firstName = "Jane";
billTo.lastName = "Smith";
request.billTo = billTo;

Card card = new Card();
card.accountNumber = "4111111111111111";
request.card = card;

// there is one item in this sample
request.item = new Item[1];
Item item = new Item();
item.id = "0";
item.unitPrice = "29.95";
request.item[0] = item;
```

The example above shows only a partial list of the fields you must send. Refer to "Related Documents," page 21 for information about the guides that list all of the fields for the services that you are requesting.

Sending the Request

You next send the request to CyberSource, store the reply in a new `ReplyMessage` object, and handle several exceptions that you might receive.

```
try {
  ReplyMessage reply = SoapClient.RunTransaction( request );
  SaveOrderState();
  // "Using the Decision and Reason Code," page 234 describes the ProcessReply
  // method.
  ProcessReply( reply );
} catch (CryptographicException ce) {
  SaveOrderState();
  Console.WriteLine( ce.ToString() );
  Console.WriteLine( sbe.ToString() );
} catch (WebException we) {
  SaveOrderState();
  /*
   * Some types of WebException indicate that the transaction may have been
   * completed by CyberSource. The sample code shows how to identify these exceptions.
   * If you receive such an exception, and your request included a payment service,
   * you should use the CyberSource transaction search screens to determine whether
   * the transaction was processed.
   */
  Console.WriteLine( we.ToString() );
}
private static void SaveOrderState() {
  /*
   * This is where you store the order state in your system for post-transaction
   * analysis. Be sure to store the consumer information, the values of the reply
   * fields, and the details of any exceptions that occurred.
   */
}
```

In the preceding example, when an exception occurs, the exception is printed to the console. Your web store should also display a message to the consumer indicating that you were unable to process the order. The sample code for the SOAP client shows you how to provide feedback to the consumer.

Also, if the transaction fails, and the request did not include any payment services, you may be able to resend the transaction. The sample code for the SOAP client shows you how to do this.

Interpreting the Reply

After the CyberSource server processes your request, it sends a reply message that contains information about the services you requested. You receive different fields depending on the services you request and the outcome of each service.

To use the reply information, you must integrate it into your system and any other system that uses that data. For example, you can store the reply information in a database and send it to other back office applications.

You must write an error handler to process the reply information that you receive from CyberSource. Do not show the reply information directly to consumers. Instead, present an appropriate response that tells consumers the result.

Important Because CyberSource may add reply fields and reason codes at any time, you should parse the reply data according to the names of the fields instead of their order in the reply.

The most important reply fields to evaluate are the following:

- **decision**: A one-word description of the results of your request. The decision is one of the following:
  - `ACCEPT` if the request succeeded
  - `REJECT` if one or more of the services in the request was declined
  - `REVIEW` if you use CyberSource Decision Manager and it flags the order for review. See "For CyberSource Advanced Merchants: Handling Decision Manager Reviews," page 258 for more information.
  - `ERROR` if there was a system error. See "Retrying When System Errors Occur," page 260 for important information about handling retries in the case of system errors.
- **reasonCode**: A numeric code that provides more specific information about the results of your request.

You also receive a reason code for each service in your request. You can use these reason codes to determine whether a specific service succeeded or failed. If a service fails, other services in your request may not run. For example, if you request a credit card

authorization and capture, and the authorization fails, the capture does not run. The reason codes for each service are described in the *Credit Card Services User Guide* for CyberSource Essentials merchants or in the service developer guide for CyberSource Advanced merchants.

Important CyberSource reserves the right to add new reason codes at any time. If your error handler receives a reason code that it does not recognize, it should use the decision to interpret the reply.

Using the Decision and Reason Code

The following example shows how you can use the decision and the reason code to display an appropriate message to the consumer.

```
private static bool ProcessReply( ReplyMessage reply ) {
  string template = GetTemplate( reply.decision.ToUpper() );
  string content  = GetContent( reply );

  // This example writes the message to the console. Choose an appropriate display
  // method for your own application.
  Console.WriteLine( template, content );
}

private static string GetTemplate( string decision ) {
  // Retrieves the text that corresponds to the decision.
  if ("ACCEPT".Equals( decision )) {
    return( "The order succeeded.{0}" );
  }
  if ("REJECT".Equals( decision )) {
    return( "Your order was not approved.{0}" );

  }

  // ERROR, or an unknown decision
  return( "Your order could not be completed at this time.{0}" +
          "\nPlease try again later." );
}
private static string GetContent( ReplyMessage reply ) {
  /*
   * Uses the reason code to retrieve more details to add to the template.
   * The messages returned in this example are meant to demonstrate how to retrieve
   * the reply fields. Your application should display user-friendly messages.
   */
  int reasonCode = int.Parse( reply.reasonCode );
  switch (reasonCode) {
    // Success
    case 100:
      return( "\nRequest ID: " + reply.requestID );
    // Missing field or fields
    case 101:
      return( "\nThe following required fields are missing: " +
              EnumerateValues( reply.missingField ) );

    // Invalid field or fields

    case 102:
      return( "\nThe following fields are invalid: " +
              EnumerateValues( reply.invalidField ) );

    // Insufficient funds
    case 204:
      return( "\nInsufficient funds in the account. Please use a " +
              "different card or select another form of payment." );
    // Add additional reason codes here that you must handle more specifically.
    default:
      // For all other reason codes, such as unrecognized reason codes or codes
      // that do not require special handling, return an empty string.
      return( String.Empty );
  }
}
private static string EnumerateValues( string[] array ) {
  System.Text.StringBuilder sb = new System.Text.StringBuilder();
  foreach (string val in array) {
    sb.Append( val + "\n" );

  }
  return( sb.ToString() );
}
```

For CyberSource Advanced Merchants: Handling Decision Manager Reviews

The information in this section applies only to CyberSource Advanced merchants.

If you use CyberSource Decision Manager, you may also receive the `REVIEW` value in the **decision** field. `REVIEW` means that Decision Manager has marked the order for review based on how you configured the Decision Manager rules.

If you will be using Decision Manager, you have to determine how to handle the new `REVIEW` value. Ideally, you will update your order management system to recognize the `REVIEW` response and handle it according to your business rules. If you cannot update your system to handle the `REVIEW` response, CyberSource recommends that you choose one of these options:

- If you authorize and capture the credit card payment at the same time, treat the `REVIEW` response like a `REJECT` response. Rejecting any orders that are marked for review may be appropriate if your product is a software download or access to a Web site. If supported by your processor, you may also want to reverse the authorization.

- If you approve the order after reviewing it, convert the order status to `ACCEPT` in your order management system. You can request the credit card capture without requesting a new authorization.

- If you approve the order after reviewing it but cannot convert the order status to `ACCEPT` in your system, request a new authorization for the order. When processing this new authorization, you must disable Decision Manager. Otherwise the order will be marked for review again. For details about the API field that disables Decision Manager, see the *Decision Manager Developer Guide Using the Simple Order API* (PDF | HTML) or the *Decision Manager Developer Guide Using the SCMP Order API* (PDF | HTML).

Alternately, you can specify a custom business rule in Decision Manager so that authorizations originating from a particular internal IP address at your company are automatically accepted.

If supported by your processor, you may want to reverse the original authorization.

Requesting Multiple Services

When you request multiple services in one request, CyberSource processes the services in a specific order. If a service fails, CyberSource does not process the subsequent services in the request.

For example, in the case of a sale (a credit card authorization and a capture requested together), if the authorization service fails, CyberSource will not process the capture service. The reply you receive only includes reply fields for the authorization.

Many CyberSource services include “ignore” fields that tell CyberSource to ignore the result from the first service when deciding whether to run the subsequent services. In the case of the sale, even though the issuing bank gives you an authorization code, CyberSource might decline the authorization based on the AVS or card verification results. Depending on your business needs, you might choose to capture these types of declined authorizations anyway. You can set the **businessRules_ignoreAVSResult** field to `"true"` in your combined authorization and capture request:

```
BusinessRules businessRules = new BusinessRules();

businessRules.ignoreAVSResult = "true";

request.businessRules = businessRules;
```

This tells CyberSource to continue processing the capture even if the AVS result causes CyberSource to decline the authorization. In this case you would then get reply fields for both the authorization and the capture in your reply.

Note

You are charged only for the services that CyberSource performs.

Retrying When System Errors Occur

You must design your transaction management system to include a way to correctly handle CyberSource system errors. Depending on which payment processor is handling the transaction, the error may indicate a valid CyberSource system error, or it may indicate a processor rejection because of some type of invalid data. In either case, CyberSource recommends that you do not design your system to retry sending a transaction many times in the case of a system error.

Instead, CyberSource recommends that you retry sending the request only two or three times with successively longer periods of time between each retry. For example, after the first system error response, wait 30 seconds and then retry sending the request. If you receive the same error a second time, wait one minute before you send the request again. Depending on the situation, you may decide you can retry sending the request after a longer time period. Determine what is most appropriate for your business situation.

If after several retry attempts you are still receiving a system error, the error may actually be caused by a processor rejection, not a CyberSource system error. In that case, we suggest one of these actions:

- Search for the transaction in the Business Center (depending on which one you normally use), look at the description of the error on the Transaction Detail page, and call your processor to determine if and why the transaction was rejected.
- Contact CyberSource Customer Support to confirm whether your error is truly caused by a CyberSource system issue.

If TSYS Acquiring Solutions is your processor, you may want to follow the first suggestion because several common TSYS Acquiring Solutions processor responses can be returned as system errors, and only TSYS Acquiring Solutions can address these errors.

Creating an Application Settings File

After you finish writing code for your integration, you must create an application settings file. This file must contain at least the directory that contains your security key and the location of the CyberSource server.

See Table 35, "Fields in the Settings File," on page 223 for a complete list of settings. You can use the settings files that come with the sample applications as a starting point for your own settings file. See "Configuring the Test Applications," page 223 for more information.

Setting the Connection Limit

This section explains how to increase the number of simultaneous connections between the client and CyberSource.

By default, you can create only two simultaneous connections to an HTTP server. By increasing the number of connections, you can avoid a backlog of requests during times of very high transaction volume. Microsoft recommends for the connection limit a value that is 12 times the number of CPUs. For example, if you have two CPUs, you can set the connection limit to 24. To determine the optimum setting for your application, make sure to run performance tests.

Examples

You can increase the number of connections in many ways, for example by using an application- or server-specific configuration file where you can change the setting for a single or for all hosts. The examples below describe briefly some of the methods that you can use to increase connection limits.

cybs.connectionLimit

When set to a value other than `-1`, the `cybs.connectionLimit` setting in the client increases the limit for the host where you are sending the request by executing these statements on your behalf:

```
ServicePoint sp = ServicePointManager.FindServicePoint(uri);
sp.ConnectionLimit = config.ConnectionLimit;
```

<connectionManagement>

You can set the connection limit by using .NET's `<connectionManagement>` tag. In this example, the connection limit for CyberSource's test and production hosts is 12 while the limit for all other hosts is 2:

```
<system.net>
  <connectionManagement>
    <add address = "https://ics2wstesta.ic3.com" maxconnection = "12" />
    <add address = "https://ics2wsa.ic3.com" maxconnection = "12" />
    <add address = "*" maxconnection = "2" />
  </connectionManagement>
</system.net>
```

DefaultConnectionLimit

You can set the connection limit for all hosts to which your application is connected before a connection is made by using the following line in your start-up code:

```
ServicePointManager.DefaultConnectionLimit = your_value_here;
```

References

For more information on these and other methods to increase the connection limits, see the following Microsoft documentation:

- Managing Connections in the *.Net Framework Developer's Guide* (`http://msdn2.microsoft.com/en-us/library/7af54za5.aspx`).

Sample ASP.NET Code Using Visual Basic

The following sample files illustrate how to use the CyberSource Name-Value Pair client in ASP.NET using Visual Basic. The `web.config` file is a sample web application configuration file containing sample entries required by the client. The other files are simple web forms and their corresponding code-behind files. The `Checkout.aspx` file contains a pre-filled form. When you press the Submit button, it will post the entered data to `Checkout2.aspx`, which will send the transaction to CyberSource.

Listing 1: web.config

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

      <add key="cybs.merchantID" value="your_merchant_id"/>
      <add key="cybs.keysDirectory" value="c:\keys"/>
      <add key="cybs.sendToProduction" value="false"/>

      <!-- Logging should normally be disabled in production as it would  -->
      <!-- slow down the processing.  Enable it only when troubleshooting -->
      <!-- an issue.                                                       -->
      <add key="cybs.enableLog" value="false"/>
      <add key="cybs.logDirectory" value="C:\Program Files\CyberSource
Corporation\simapi-net-2.0-5.0.0\logs"/>

      <!-- Please refer to the Connection Limit section in the README for -->
      <!-- details on this setting and alternate ways to set the          -->
      <!-- connection limit.  When not specified or is set to -1, the     -->
      <!-- client will implicitly use the connection limit currently in   -->
      <!-- force, which would be 2 if none of the alternate methods are   -->
      <!-- used.                                                           -->
      <add key="cybs.connectionLimit" value="-1"/>

   </appSettings>
</configuration>
```

Listing 2: Checkout.aspx

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Checkout.aspx.vb"
Inherits="NVP" Debug="true"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
   <title>Name Value Pair - Order Page</title>
</head>
<body>
<form action="Checkout2.aspx" method="post">
Please confirm the information below and click the Submit button to perform the
authorization.
<br/>
   <h3>Billing Information</h3>
   First Name:<br/>
   <input type="text" name="billTo_firstName" value="John"/>
   <br/>Last Name:<br/>
   <input type="text" name="billTo_lastname" value="Doe"/>
   <br/>Street Address:<br/>
   <input type="text" name="billTo_street1" value="1295 Charleston Road"/>
   <br/>City:<br/>
   <input type="text" name="billTo_city" value="Mountain View"/>
   <br/>State:<br/>
   <input type="text" name="billTo_state" value="CA"/>
   <br/>Postal Code:<br/>
   <input type="text" name="billTo_postalCode" value="94043"/>
   <br/>Country:<br/>
   <input type="text" name="billTo_country" value="US"/>
   <br/>
   <h3>Credit Card Information</h3>
   Amount:<br/>
   <input type="text" name="item_0_unitPrice" value="10.00"/>
   <br/>Credit Card Number:<br/>
   <input type="text" name="card_accountNumber" value="4111111111111111"/>
   <br/>Expiration month (mm):<br/>
   <input type="text" name="card_expirationMonth" value="12"/>
   <br/>Expiration year (yyyy):<br/>
   <input type="text" name="card_expirationYear" value="2010"/>
   <br/>Email Address:<br/>
   <input type="text" name="billTo_email" value="nobody@cybersource.com"/>
   <br/><input type="submit" value="Submit"/>
</form>
</body>
</html>
```

Listing 3: Checkout.aspx.vb

```
Partial Class NVP
   Inherits System.Web.UI.Page
End Class
```

Listing 4: Checkout2.aspx

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Checkout2.aspx.vb"
Inherits="NVP2" Debug="true"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
   <title>Name Value Pair - Receipt</title>
</head>
<body>
   <form id="form1" runat="server">
   <div>

   </div>
   </form>
</body>
</html>
```

Listing 5: Checkout2.aspx.vb

```
Imports CyberSource.Clients.NVPClient

Partial Class NVP2
   Inherits System.Web.UI.Page

   Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
      'Declare the request hashtable
      Dim oRequest As New Hashtable

      'Add non-user input fields
      oRequest.Add("ccAuthService_run", "true")
      oRequest.Add("merchantReferenceCode", "MRC-5254555")

      'Add user input fields from post
      oRequest.Add("billTo_firstName", Request.Form("billTo_firstName"))
      oRequest.Add("billTo_lastName", Request.Form("billTo_lastName"))
      oRequest.Add("billTo_street1", Request.Form("billTo_street1"))
      oRequest.Add("billTo_city", Request.Form("billTo_city"))
      oRequest.Add("billTo_state", Request.Form("billTo_state"))
      oRequest.Add("billTo_postalCode", Request.Form("billTo_postalCode"))
      oRequest.Add("billTo_country", Request.Form("billTo_country"))
      oRequest.Add("billTo_email", Request.Form("billTo_email"))
      oRequest.Add("card_accountNumber", Request.Form("card_accountNumber"))
      oRequest.Add("card_expirationMonth", Request.Form("card_expirationMonth"))
      oRequest.Add("card_expirationYear", Request.Form("card_expirationYear"))
      oRequest.Add("item_0_unitPrice", Request.Form("item_0_unitPrice"))
      oRequest.Add("purchaseTotals_currency", "USD")

      'Declare the reply hashtable
      Dim varReply As New Hashtable

      'Run the transaction
      varReply = CyberSource.Clients.NVPClient.RunTransaction(oRequest)

      'Print reply data to the browser
      Response.Write("reasonCode: " & varReply("reasonCode").ToString)
      Response.Write("<BR>Decision: " & varReply("decision").ToString)
      Response.Write("<BR>RequestID: " & varReply("requestID").ToString)
      Response.Write("<BR>Merchant Reference Code: " &
varReply("merchantReferenceCode").ToString)
   End Sub

End Class
```

CHAPTER 7

Java Client

Important

- The Java client for the Simple Order API is supported on 32-bit operating systems only.
- If you are building an application to sell to others, see Appendix A, "Using the Client Application Fields," on page 390. This appendix has a list of API fields you can use in your request that describe the application, its version, and its user. If you use these fields in your request, you can view their values in the Transaction Search Details window of the Business Center.

Choosing Your API and Client

API Variations

Choose either of these options of the Simple Order API:

- Name-value pairs: simpler to use. The test that you run immediately after installing the client uses name-value pairs.
- XML: requires you to create and parse XML documents

To introduce new API fields and features, CyberSource regularly updates the Simple Order API. You can update your existing client to work with the new API version. For the latest version of the server-side API for the CyberSource services, go to https://ics2wsa.ic3.com/commerce/1.x/transactionProcessor. When configuring the client, indicate the version of the API (not the current version of the client) you want to use in the targetAPIVersion configuration property. For example, to use the 1.18 version of the API, set the property to `1.18`. For more information, see "targetAPIVersion," page 273.

Client Versions

The client version is the version of the client-side code that you use to access the CyberSource services. This version is different from the API version.

A direct upgrade path from the 1.5.0 version of the Web Services Client for Java to the most recent version of the client is not available because the client was redesigned starting with the 2.0.0 release.

Sample Code

The client package contains two samples that you can use to test the client:

- Name-value pairs: See `AuthCaptureSample.java` in *`<main directory>`*`/samples/nvp/src/com/cybersource/sample`.
- XML: Before implementing your code to process XML requests, CyberSource recommends that you examine the name-value pair sample code listed above.

For the XML sample code, see `AuthSample.java` in *`<main directory>`*`/samples/xml/src/com/cybersource/sample`.

Basic Java Program Example

The example below shows the primary code required to send a Simple Order API request for credit card authorization and process the reply. The example uses name-value pairs. For a complete example, see the sample program included in the package (see "Sample Code," page 268). "Using Name-Value Pairs," page 277 shows you how to create the code.

```
package com.cybersource.sample;
import java.util.*;
import com.cybersource.ws.client.*;

public class SimpleAuthSample

{

  public static void main( String[] args )

  {
    Properties props = Utility.readProperties( args );
    HashMap request = new HashMap();

    // In this sample, we are processing a credit card authorization.
    request.put( "ccAuthService_run", "true" );

    // Add required fields
    request.put( "merchantReferenceCode", "MRC-14344" );
    request.put( "billTo_firstName", "Jane" );
    request.put( "billTo_lastName", "Smith" );
    request.put( "billTo_street1", "1295 Charleston Road" );
    request.put( "billTo_city", "Mountain View" );
    request.put( "billTo_state", "CA" );
    request.put( "billTo_postalCode", "94043" );
    request.put( "billTo_country", "US" );
    request.put( "billTo_email", "jsmith@example.com" );
    request.put( "card_accountNumber", "4111111111111111" );
    request.put( "card_expirationMonth", "12" );
    request.put( "card_expirationYear", "2010" );
    request.put( "purchaseTotals_currency", "USD" );
    // This sample order contains two line items.
    request.put( "item_0_unitPrice", "12.34" );
    request.put( "item_1_unitPrice", "56.78" );

    // Add optional fields here according to your business needs.
    // For information about processing the reply,
    // see "Using the Decision and Reason Code Fields," page 282.
    try

    {
        HashMap reply = Client.runTransaction( request, props );
    }

    catch (ClientException e) {

        if (e.isCritical())
        {
            handleCriticalException( e, request );
        }
    }
    catch (FaultException e) {
        if (e.isCritical())
        {
            handleCriticalException( e, request );
        }
    }
  }
}
```

Installing and Testing the Client

Minimum System Requirements

- This client is supported on the Windows 2000/XP/2003, Linux, and Solaris platforms.
- The minimum Java SDK supported are Oracle or IBM Java SDK 1.2 or later. Depending on the package that you choose, you also need one of these:
    - For Oracle Java SDK versions earlier than 1.4.0, you need the Java Secure Socket Extension (JSSE) 1.0.3_02 or later (see http://java.sun.com/products/jsse).
    - For IBM Java SDK, you need IBMJSEE 1.0.2 or later.

Transaction Security Keys

The first thing you must do is create your security key. The client uses the security key to add a digital signature to every request that you send. This signature helps ensure that no one else can use your CyberSource account to process orders. You specify the location of your key when you configure the client.

Important

You must generate two transaction security keys—one for the CyberSource production environment and one for the test environment. For information about generating and using security keys, see *Creating and Using Security Keys* (PDF | HTML).

Warning

You must protect your security key to ensure that your CyberSource account is not compromised.

Installing the Client

To install the client:

Step 1 Create a target directory for the client.

Step 2 Download the latest version of the client.

The package covers Windows, Linux, and Solaris.

Step 3 Unzip the package to your target directory.

<target directory>`/simapi-java-`***`n.n.n`***
where ***n.n.n*** represents the version of the client package.

Step 4 To configure the client, see "Configuring Client Properties," page 272 below.

Step 5 To test the client, see "Testing the Client," page 275.

Step 6 When done, see "Going Live," page 276.

Step 7 Create your own code for requesting CyberSource services by following either "Using Name-Value Pairs," page 277 or "Using XML," page 286.

Configuring Client Properties

The client requires certain properties to run transactions. The samples provided in the *<main directory>*`/samples/nvp` and *<main directory>*`/samples/xml` folders read a file called `cybs.properties` into a Properties object which is passed to the runTransaction() method. Table 36, "Configuration Properties," on page 273 describes the properties that you can set. Note that the default `cybs.properties` file that comes with the client package does not include all of the properties listed in the table. It includes only the ones required to run the sample.

The client also includes additional property configuration capabilities. For example, you can configure for multiple merchants or configure using system properties. For more information, see "Advanced Configuration Information," page 296.

Note For Java SDK 1.4.x, the client sets the system properties `https.proxyHost` and `https.proxyPort` to the values of the client properties `proxyHost` and `proxyPort`. If these system properties are defined beforehand, possibly by using the `-D` option in the command line, the system properties will take precedence.

Table 36 Configuration Properties

Property	Description
`merchantID`	This client uses this value if you do not specify a merchant ID in the request itself. This value is case sensitive.
`keysDirectory`	Location of the merchant's security keys. Although UNC paths are allowed, for faster request processing, CyberSource recommends that you store your key locally. You must use forward slashes even in a Windows environment (for example: `c:/keys)`. The client includes a `keys` directory that you can use.
`sendToProduction`	Flag that indicates whether the transactions for this merchant should be sent to the production server. Use one of these values: ■ `false`: Send to the test server. (default setting) ■ `true`: Send to the production server
`targetAPIVersion`	Version of the Simple Order API to use, such as `1.18`. For the list of available versions, go to https://ics2wsa.ic3.com/commerce/1.x/transactionProcessor. Changes in each version are described in the *Simple Order API Release Notes*. Do not set this property to the current version of the client. See "Client Versions," page 268 for more information.
`keyFilename`	Name of the security key file name in the format `<`***`security_key_name`***`>.p12`.
`serverURL`	Alternative server URL to use. For more information, see "Using Alternate Server Properties," page 296. Give the complete URL because it will be used exactly as specified here.
`namespaceURI`	Alternative namespace URI to use. Give the complete namespace URI because it will be used exactly as specified here. For more information, see "Using Alternate Server Properties," page 296.
`enableLog`	Flag directing the client to log transactions and errors. Use one of these values: ■ `false`: Do not enable logging (default setting) ■ `true`: Enable logging PCI **Important** Logging can cause very large log files to accumulate. Therefore, CyberSource recommends that you use logging only when troubleshooting problems. To comply with all Payment Card Industry (PCI) and Payment Application (PA) Data Security Standards regarding the storage of credit card and card verification number data, the logs that are generated contain only masked credit card and card verification number data (CVV, CVC2, CVV2, CID, CVN). Follow these guidelines: ■ Use debugging temporarily for diagnostic purposes only. ■ If possible, use debugging only with test credit card numbers. ■ Never store clear text card verification numbers. ■ Delete the log files as soon as you no longer need them. ■ Never send email to CyberSource containing personal and account information, such as customers' names, addresses, card or check account numbers, and card verification numbers. For more information about PCI and PABP requirements, see www.visa.com/cisp.

Table 36 Configuration Properties (Continued)

Property	Description
`logDirectory`	Directory to which to write the log file. UNC paths are allowed. You must use forward slashes even in a Windows environment, for example: `c:/logs`. The client does not create this directory; instead you must specify an existing directory. The client includes a `logs` directory that you can use.
`logFilename`	Log file name. The client uses `cybs.log` by default.
`logMaximumSize`	Maximum size in megabytes for the log file. The default value is `"10"`. When the log file reaches the specified size, it is archived into `cybs.log`.<***yyyymmdd***`T`***hhmmssxxx***> and a new log file is started. The ***xxx*** indicates milliseconds.
`timeout`	**Important** Ignore this property. Instead set a specific amount of time that is acceptable to your business. Number of seconds to wait for reply before timing out. Default value is 130. This property does not have an effect if useHttpClient is `false` and you are using `cybsclients14.jar`.
`useHttpClient`	Flag directing the client to use Apache HttpClient for the HTTPS communication. Use one of these values: ▪ `false`: (default setting) Do not use Apache HttpClient. Use built-in HttpURLConnection. The timeout property does not have an effect if useHttpClient is `false` and you are using `cybsclients14.jar`. ▪ `true`: Use Apache HttpClient. When useHttpClient is `true`, your CLASSPATH must include the three `commons-*.jar` files shipped with the package.
`proxyHost`	Optional host name or IP address of the HTTP proxy server.
`proxyPort`	Port number of the proxy server. The default is 8080. This property is ignored if you do not specify `proxyHost`.
`proxyUser`	User name used to authenticate against the proxy server if required.
`proxyPassword`	Password used to authenticate against the proxy server if required.

Testing the Client

After you install and configure the client, test it to ensure the installation was successful.

To test the client:

Step 1 If you are using the Solaris or Linux platform, set the execute permission on the `runSample.sh` script, for example: **`chmod 755 runSample.sh`**

Step 2 If you are using Java SDK 1.5 or later, replace `cybsclients14.jar` with `cybsclients15.jar` in `runSample.sh`.

Step 3 At a command prompt, type this line:

Windows	**`runSample.bat`**
Unix or Linux	**`runSample.sh`**

If `JAVA_HOME` is defined, the script uses `<JAVA_HOME>/bin/java`. Otherwise, it uses whatever java is in the path.

If the client is installed correctly, the requests and replies for a credit card authorization and a follow-on capture appear.

If the client is not installed correctly, a fault exception appears:

- Configuration exception if the keys directory in the `cybs.properties` file is incorrect.
- `java.net.MalformedURLException: unknown protocol: https`: see "Resolving Connection Issues," page 297.
- `javax.net.ssl.SSLException untrusted server cert chain`: see "Importing the Root CA Certificate," page 299.

Going Live

When you finish configuring and testing the client, your deployment is ready to go live.

Important Make sure that your client is set to send transactions to the production server, not the test server. See the description of "sendToProduction," page 273.

CyberSource Essentials Merchants

If you use CyberSource Essentials services, you can use the Business Center site to go live. For a description of the process of going live, see the “Steps for Getting Started” section in *Getting Started with CyberSource Essentials*.

Important You must also configure your client so that it sends transactions to the production server and not the test server. See the description of the `sendToProduction` property in Table 36, "Configuration Properties," on page 273.

After your deployment goes live, use real card numbers and other data to test every card type you support. Because these are real transactions in which you are buying from yourself, use small monetary amounts to do the tests. Process an authorization, then capture the authorization, and later refund the money. Use your bank statements to verify that money is deposited into and withdrawn from your merchant bank account as expected. If you have more than one CyberSource merchant ID, test each one separately.

CyberSource Advanced Merchants

If you use CyberSource Advanced services, see the “Steps for Getting Started” chapter in *Getting Started with CyberSource Advanced* for information about going live.

When your deployment goes live, your CyberSource account is updated so that you can send transactions to the CyberSource production server. If you have not already done so, you must provide your banking information to CyberSource so that your processor can deposit funds to your merchant bank account.

After CyberSource confirms that your account is live, make sure that you update your system so that it can send requests to the production server (`ics2wsa.ic3.com`) using your security keys for the production environment. The test server (`ics2wstesta.ic3.com`) cannot be used for real transactions. For more information about sending transactions to the production server. see the description of the configuration property "sendToProduction," page 273.

After your deployment goes live, use real card numbers and other data to test every card type, currency, and CyberSource application that your integration supports. Because these are real transactions in which you are buying from yourself, use small monetary amounts to do the tests. Use your bank statements to verify that money is deposited into and withdrawn from your merchant bank account as expected. If you have more than one CyberSource merchant ID, test each one separately.

Using Name-Value Pairs

This section explains how to write Java programs that request CyberSource services by using name-value pairs.

Requesting CyberSource Services

To request CyberSource services, write code that can perform these actions:

- Collect information for the CyberSource services that you will use
- Assemble the order information into requests
- Send the requests to the CyberSource server

Important

The CyberSource servers do not support persistent HTTP connections.

- Process the reply information

For the list of API fields that you must add to your requests and will see in the replies, use the guide that describes the service. See "Related Documents," page 21.

The code in this section's example is incomplete. For a complete sample program, see the `AuthCaptureSample.java` file in *`<main directory>`*`/samples/nvp/src/com/cybersource/sample` directory.

Note

If you make any changes to the `AuthCaptureSample.java` sample, you must rebuild the sample before using it. Use the `compileSample` batch file or shell script provided in the `sample` directory.

If you use Java SDK 1.5 or later, replace `cybsclients14.jar` with `cybsclients15.jar` in the `compileSample` script.

Creating and Sending Requests

To use any CyberSource service, you must create and send a request that includes the required information for that service. The following example shows basic code for requesting a credit card authorization. In this example, Jane Smith is buying an item for $29.95.

Importing the Client Classes

Add the following import statements:

```
import java.util.*;
import com.cybersource.ws.client.*;
```

Depending on your application, you might need to add more import statements.

Loading the Configuration File

Load the configuration file:

```
Properties props = Utility.readProperties( args );
```

The sample reads the configuration settings from the properties file specified in the command line. If you do not specify a file, the sample looks for the file `cybs.properties` in the current directory.

Creating an Empty Request

Create a hashtable that holds the request fields:

```
HashMap request = new HashMap();
```

Adding Services to the Request

Indicate the service that you want to use by adding a field to the request, such as a credit card authorization:

```
request.put( "ccAuthService_run", "true" );
```

You can request multiple services by adding additional fields to the request. When you request multiple services in one request, CyberSource processes the services in a specific order. If a service fails, CyberSource does not process the subsequent services in the request. You are charged only for the services that CyberSource performs.

Example **For All Merchants: Requesting Multiple Services**

For example, if you fulfill the order immediately, you can request a credit card authorization and capture together, called a *sale.* If the authorization service fails, CyberSource does not process the capture service. The reply you receive includes reply fields only for the authorization:

```
request.put( "ccAuthService_run", "true" );
request.put( "ccCaptureService_run", "true" );
```

Example **For Merchants Using CyberSource Advanced Services: Requesting Multiple Services**

Many CyberSource services include fields that tell CyberSource to ignore the result from the first service when deciding whether to run the subsequent services. In the case of the sale, even though the issuing bank gives you an authorization code, CyberSource may decline the authorization based on the AVS or card verification results. Depending on your business needs, you might choose to capture these declined authorizations. To do so, in your combined authorization and capture request, set the **businessRules_ ignoreAVSResult** field to true:

```
request.put( "businessRules_ignoreAVSResult", "true" );
```

This line tells CyberSource to process the capture even if the AVS result causes CyberSource to decline the authorization. In this case, the reply would contain fields for the authorization and the capture.

Adding Service-Specific Fields to the Request

Add the fields that are used by the services you are requesting. If you request multiple services that share fields, add the field only once.

```
request.put( "billTo_firstName", "Jane" );
request.put( "billTo_lastName", "Smith" );
request.put( "card_accountNumber", "4111111111111111" );
request.put( "item_0_unitPrice", "29.95" );
```

The example above shows only a partial list of the fields you must send. The developer guides for the service you are using contains a complete list of API request and reply fields available for that service.

Sending the Request

Send the request to CyberSource, store the reply in a new hashtable, and interpret the exceptions that you might receive:

```
try {
  HashMap reply = Client.runTransaction( request, props );
  //"Using the Decision and Reason Code Fields," page 282 illustrates how you
  //might design a ProcessReply() method to handle the reply.
  processReply( reply );
}
catch (FaultException e)
{
  System.out.println( e.getLogString() );
}
catch (ClientException e)
{
  System.out.println( e.getLogString() );
}
```

In the example above, when an exception occurs, the exception is printed to the console. Your web store should also display to the customer a message indicating that you were unable to process the order. "Using the Decision and Reason Code Fields," page 282 shows how to provide feedback to the customer.

Interpreting Replies

After your request is processed by the CyberSource server, it sends a reply message that contains information about the services you requested. You receive fields relevant to the services that you requested and to the outcome of each service.

To use the reply information, you must integrate it into your system and any other system that uses that data. For example, you can store the reply information in a database and send it to other back office applications.

You must write an error handler to process the reply information that you receive from CyberSource. Do not show the reply information directly to customers. Instead, present an appropriate response that tells customers the result.

Important CyberSource may add reply fields and reason codes at any time. If your error handler receives a reason code that it does not recognize, it should use the **decision** to interpret the reply. Parse the reply data according to the names of the fields instead of their order in the reply.

These are the most important reply fields:

- **decision**: A one-word description of the results of your request. The possible values are as follows:
  - `ACCEPT` if the request succeeded.
  - `REJECT` if one or more of the services in the request was declined.
  - `REVIEW` (Advanced package only) if you use Decision Manager, and the order is marked for review. For more information, see "Handling Decision Manager Reviews (CyberSource Advanced Services Only)," page 284.
  - `ERROR` if a system error occurred. For more information, see "Handling System Errors," page 284.
- **reasonCode**: A numeric code that provides more specific information about the results of your request.

You also receive a reason code for each service in your request. You can use these reason codes to determine whether a specific service succeeded or failed. If a service fails, other services in your request may not run. For example, if you request a credit card authorization and capture, and the authorization fails, the capture does not run. The reason codes for each service are described in the *Credit Card Services User Guide* for CyberSource Essentials merchants or in the service developer guide for CyberSource Advanced merchants

Using the Decision and Reason Code Fields

This example shows how you can use the decision and the reason code to display an appropriate message to the customer.

Note The processReply() method described below is not included in the sample code in the client package.

```
private static boolean processReply( HashMap reply )
  throws ClientException {
    MessageFormat template = new MessageFormat(
    getTemplate( (String) reply.get( "decision" ) ) );
    Object[] content = { getContent( reply ) };
    /*
     * This example writes the message to the console. Choose an appropriate display
     * method for your own application.
     */
    System.out.println( template.format( content ) );
}

private static String getTemplate( String decision ) {
  // Retrieves the text that corresponds to the decision.
  if ("ACCEPT".equalsIgnoreCase( decision )) {
    return( "Your order was approved.{0}" );
  }
  if ("REJECT".equalsIgnoreCase( decision )) {
    return( "Your order was not approved.{0}" );
  }

  // ERROR
  return( "Your order cannot be completed at this time.{0}" +
          "\nPlease try again later." );
}
private static String getContent( HashMap reply )
  throws ClientException {
  /*
   * Uses the reason code to retrieve more details to add to the template.
   * The strings returned in this sample are meant to demonstrate how to retrieve
   * the reply fields. Your application should display user-friendly messages.
   */
  int reasonCode =
    Integer.parseInt( (String) reply.get( "reasonCode" ) );
  switch (reasonCode) {
    // Success
    case 100:
      return( "\nRequest ID: " + (String) reply.get( "requestID" );
    // Missing field or fields
    case 101:
      return( "\nThe following required field(s) are missing:\n" +
              enumerateValues( reply, "missingField" ) );

    // Invalid field or fields
    case 102:
      return( "\nThe following field(s) are invalid:\n" +
               enumerateValues( reply, "invalidField" ) );
    // Insufficient funds
    case 204:
      return( "\nInsufficient funds in the account. Please use a different " +
              "card or select another form of payment." );
    // Add additional reason codes here that you must handle specifically.
    default:
      // For all other reason codes (for example, unrecognized reason codes, or
      // codes that do not require special handling), return an empty string.
      return( "" );
  }
}

private static String enumerateValues( Map reply, String fieldName ) {
  StringBuffer sb = new StringBuffer();
  String key, val = "";
  for (int i = 0; ; ++i) {
    key = fieldName + "_" + i;
    if (!reply.containsKey( key )) {
      break;
    }
    val = (String) reply.get( key );
    if (val != null) {
      sb.append( val + "\n" );
    }
  }
  return( sb.toString() );
}
```

Handling Decision Manager Reviews (CyberSource Advanced Services Only)

If you use CyberSource Decision Manager, you may also receive the `REVIEW` value in the **decision** field. `REVIEW` means that Decision Manager has marked the order for review based on how you configured the Decision Manager rules.

If you will be using Decision Manager, you have to determine how to handle the new `REVIEW` value. Ideally, you will update your order management system to recognize the `REVIEW` response and handle it according to your business rules. If you cannot update your system to handle the `REVIEW` response, CyberSource recommends that you choose one of these options:

- If you authorize and capture the credit card payment at the same time, treat the `REVIEW` response like a `REJECT` response. Rejecting any orders that are marked for review may be appropriate if your product is a software download or access to a Web site. If supported by your processor, you may also want to reverse the authorization.
- If you approve the order after reviewing it, convert the order status to `ACCEPT` in your order management system. You can request the credit card capture without requesting a new authorization.
- If you approve the order after reviewing it but cannot convert the order status to `ACCEPT` in your system, request a new authorization for the order. When processing this new authorization, you must disable Decision Manager. Otherwise the order will be marked for review again. For details about the API field that disables Decision Manager, see the *Decision Manager Developer Guide Using the Simple Order API* (PDF | HTML) or the *Decision Manager Developer Guide Using the SCMP Order API* (PDF | HTML).

Alternately, you can specify a custom business rule in Decision Manager so that authorizations originating from a particular internal IP address at your company are automatically accepted.

If supported by your processor, you may want to reverse the original authorization.

Handling System Errors

You must design your transaction management system to correctly handle CyberSource system errors, which occur when you successfully receive a reply, but the **decision** field is ERROR. For more information about the **decision**, see "Interpreting Replies," page 280. The error may indicate a valid CyberSource system error or a payment processor rejection because of invalid data.

Offline Transactions

CyberSource recommends that you resend the request two or three times only, waiting a longer period of time between each attempt. Determine what is most appropriate for your business situation.

Example Handling System Errors for Offline Transactions

After the first system error response, wait a short period of time, perhaps 30 seconds, before resending the request. If you receive the same error a second time, wait a longer period of time, perhaps 1 minute, before resending the request. If you receive the same error a third time, you may decide to try again after a longer period of time, perhaps 2 minutes.

If you are still receiving a system error after several attempts, the error may be caused by a processor rejection instead of a CyberSource system error. In this case, CyberSource recommends one of these options:

- Find the transaction in the Business Center. After looking at the description of the error on the transaction details page, call your processor to determine if and why the transaction was rejected. If your processor is TSYS Acquiring Solutions, you may want to follow this option because this processor can return several system errors that only it can address.
- Contact CyberSource Customer Support to determine whether the error is caused by a CyberSource system issue.

Online Transactions

For online transactions, inform the customer that an error occurred and request that the customer attempts to resubmit the order.

Using XML

This section explains how to write Java programs that request CyberSource services by using XML.

Requesting CyberSource Services

To request CyberSource services, write code that can perform these actions:

- Collect information for the CyberSource services that you will use
- Assemble the order information into requests
- Send the requests to the CyberSource server

Important The CyberSource servers do not support persistent HTTP connections.

- Process the reply information

For the list of API fields that you must add to your requests and will see in the replies, use the guide that describes the service. See "Related Documents," page 21.

To understand how to request CyberSource services, CyberSource recommends that you examine the name-value pair sample code provided in `AuthCaptureSample.java` before implementing your code to process XML requests. The sample code file is located in the `<main directory>/samples/nvp/src/com/cybersource/sample` directory.

The code in this section's example is incomplete. For a complete sample program, see the `AuthSample.java` file in the `<main directory>/samples/xml/src/com/cybersource/sample` directory.

Note If you make changes to the `AuthSample.java` sample, you must rebuild the sample before using it by using the `compileSample` batch file or shell script provided in the `xmlsample` directory.

If you use Java SDK 1.5 or later, replace `cybsclients14.jar` with `cybsclients15.jar` in the `compileSample` script.

Creating Requests

The client enables you to create an XML request document by using any application and sending the request to CyberSource. For example, if you have a customer relationship management (CRM) application that uses XML to communicate with other applications, you can use your CRM to generate request documents.

You must validate the request document against the XML schema for CyberSource transactions. To view the schema, look at the `xsd` file for your version of the Simple Order API.

Important If the elements in your document do not appear in the correct order, your document will not be validated, and your request will fail.

The following example, from creating an empty request to adding service-specific fields, shows a basic XML document for requesting a credit card authorization. In this example, Jane Smith is buying an item for $29.95. The XML document in this example is incomplete. For a complete example, see the `auth.xml` file in the `samples/xml` directory.

Creating an Empty Request

Start with the XML declaration and the root element:

```
<?xml version="1.0" encoding="utf-8"?>
<requestMessage xmlns="urn:schemas-cybersource-com:transaction-data-1.18">
</requestMessage>
```

When you construct a request, indicate the namespace for the elements. The namespace must use the same API version that you specify in the configuration settings.

Example **API version:** `targetAPIVersion`**=**`1.18`

Namespace: `urn:schemas-cybersource-com:transaction-data-1.18`

Adding Services to the Request

Add the services that you want to use by creating an element for that service and setting the element's `run` attribute to `true`. This example shows a credit card authorization:

```
<?xml version="1.0" encoding="utf-8"?>
<requestMessage xmlns="urn:schemas-cybersource-com:transaction-data-
1.18">
  <ccAuthService run="true"/>
</requestMessage>
```

You can request multiple services by creating additional elements. When you request multiple services in one request, CyberSource processes the services in a specific order. If a service fails, CyberSource does not process the subsequent services in the request. You are charged only for the services that CyberSource performs.

Example **For All Merchants: Requesting Multiple Services in a Request**

If you fulfill orders immediately, you can request a credit card authorization and capture together, called a sale. If the authorization service fails, CyberSource does not process the capture service. The reply that you receive contains only authorization reply fields:

```
<?xml version="1.0" encoding="utf-8"?>
<requestMessage xmlns="urn:schemas-cybersource-com:transaction-data-
1.18">
  <ccAuthService run="true"/>
  <ccCaptureService run="true"/>
</requestMessage>
```

Example **Only for Merchants Using CyberSource Advanced Services: Requesting Multiple Services in a Request**

Many CyberSource services use fields that tell CyberSource to ignore the result from the first service when deciding whether to run the subsequent services. In the case of the sale, even though the issuing bank gives you an authorization code, CyberSource may decline the authorization based on the address or card verification results. Depending on your business needs, you might choose to capture these declined authorizations. To do so, in your combined authorization and capture request, you must set the **businessRules_ignoreAVSResult** field to true:

```
<businessRules>
    <ignoreAVSResult>true</ignoreAVSResult>
</businessRules>
```

These lines tell CyberSource to process the capture even if the address verification result causes CyberSource to decline the authorization. In this case, the reply would contain fields for the authorization and the capture.

Adding Service-Specific Fields to the Request

Add the fields that are used by the services you are requesting. Most fields are child elements of container elements. For example, a `<card>` element contains the customer's credit card information. This example shows a partial list of possible fields. The developer guides for the service you are using contains a complete list of API request and reply fields for that service.

```
<?xml version="1.0" encoding="utf-8"?>
<requestMessage xmlns="urn:schemas-cybersource-com:transaction-data-
1.18">
  <billTo>
    <firstName>Jane</firstName>
    <lastName>Smith</lastName>
  </billTo>
  <item id="0">
    <unitPrice>29.95</unitPrice>
  </item>
  <card>
    <accountNumber>4111111111111111</accountNumber>
  </card>
  <ccAuthService run="true"/>
</requestMessage>
```

Sending Requests

Once you have created an XML request document, you can use Java to send the request to CyberSource.

Importing the Client Classes

Add the following import statements:

```
import java.io.*;
import java.util.*;
import javax.xml.parsers.*;
import org.w3c.dom.*;
import org.xml.sax.*;
import com.cybersource.ws.client.*;
```

Depending on your application, you might need to add more import statements.

Loading the Configuration File

Load the configuration file:

```
Properties props = Utility.readProperties( args );
```

The sample reads the configuration settings from the properties file specified in the command line. If you do not specify a file, the sample looks for the file `cybs.properties` in the current directory.

Sending the Request

Send the request to CyberSource, store the reply in a new `Document` object, and interpret the exceptions that you might receive:

```
try {
  Document request = readRequest( props, args );
  // The sample reads the files specified in the command line, or if no files are
  // specified, the sample looks for cybs.properties and auth.xml in the current
  // directory.
  Document reply = XMLClient.runTransaction( request, props );
  // "Using the Decision and Reason Code Fields," page 282 illustrates how you might
  // design a ProcessReply() method to handle the reply.
  processReply( reply );
}
catch (FaultException e)
{
  e.printStackTrace();
  System.out.println( e.getLogString() );
}
catch (ClientException e)
{
  e.getInnerException().printStackTrace();
  System.out.println( e.getLogString() );
}
```

In the preceding example, when an exception occurs, the exception is printed to the console. Your web store should also display a message to the customer indicating that you were unable to process the order. "Using the Decision and Reason Code Fields," page 282 shows how to provide feedback to the customer.

Interpreting Replies

Note The XML document that you receive in the reply always uses a prefix of `c:`, for example: `xmlns:c="urn:schemas-cybersource-com:transaction-data-1.18"`. Make sure you use an XML parser that supports namespaces.

After your request is processed by the CyberSource server, it sends a reply message that contains information about the services you requested. You receive fields relevant to the services that you requested and to the outcome of each service.

To use the reply information, you must integrate it into your system and any other system that uses that data. For example, you can store the reply information in a database and send it to other back office applications.

You must write an error handler to process the reply information that you receive from CyberSource. Do not show the reply information directly to customers. Instead, present an appropriate response that tells customers the result.

Important CyberSource may add reply fields and reason codes at any time. If your error handler receives a reason code that it does not recognize, it should use the **decision** to interpret the reply. Parse the reply data according to the names of the fields instead of their order in the reply.

These are the most important reply fields:

- **decision**: A one-word description of the results of your request:
  - `ACCEPT` if the request succeeded.
  - `REJECT` if one or more of the services in the request was declined.
  - `REVIEW` (Advanced package only) if you use Decision Manager, and the order is marked for review. For more information, see "Handling Decision Manager Reviews (CyberSource Advanced Merchants)," page 294.
  - `ERROR` if a system error occurred. For more information, see "Handling System Errors," page 294.
- **reasonCode**: A numeric code that provides more specific information about the results of your request.

You also receive a reason code for each service in your request. You can use these reason codes to determine whether a specific service succeeded or failed. If a service fails, other services in your request may not run. For example, if you request a credit card authorization and capture, and the authorization fails, the capture does not run. The reason codes for each service are described in the *Credit Card Services User Guide* for CyberSource Essentials merchants or in the service developer guide for CyberSource Advanced merchants.

Using the Decision and Reason Code

This example shows how you can use the decision and the reason code to display an appropriate message to the customer.

Note The processReply() method described below is not included in the sample code in the client package.

```
private static boolean processReply( Document reply )
  throws ClientException {
  // The following code allows you to use XPath with the CyberSource schema, which
  // uses a non-empty default namespace.
  XPathAPI xp = new XPathAPI();
  Element nsNode = reply.createElement( "nsNode" );
  // The version number (1.20) at the end of the namespaceURI below is an example.
  // Change it to the version of the API that you are using.
  nsNode.setAttribute("xmlns:cybs", "urn:schemas-cybersource-com:transaction-data
    -1.20" );
  Node replyMessage =
    getNode( xp, reply, "cybs:replyMessage", nsNode );
  String decision =
    getText( xp, replyMessage, "cybs:decision", nsNode );
  MessageFormat template =
    new MessageFormat( getTemplate( decision ) );
  Object[] content = { getContent( xp, replyMessage, nsNode ) };
  /*
   * This example writes the message to the console. Choose an appropriate display
   * method for your own application.
   */
  System.out.println( template.format( content ) );
}
  private static String getTemplate( String decision ){
    // Retrieves the text that corresponds to the decision.
    if ("ACCEPT".equalsIgnoreCase( decision )) {
      return( "Your order was approved.{0}" );
    }

    if ("REJECT".equalsIgnoreCase( decision )) {
      return( "Your order was not approved.{0}" );
    }

    // ERROR, or unknown decision
    return( "Your order cannot be completed at this time.{0}" +
            "\nPlease try again later." );
}
```

```
private static String getContent(
  XPathAPI xp, Node ctxNode, Node nsNode )
  throws XMLClientException {
  /*
   * Uses the reason code to retrieve more details to add to the template.
   * The strings returned in this sample are meant to demonstrate how to retrieve
   * the reply fields. Your application should display user-friendly messages.
   */
  int reasonCode = Integer.parseInt(
    getText( xp, ctxNode, "cybs:reasonCode", nsNode ) );
  switch (reasonCode) {
    // Success
    case 100:
      return ( "\nRequest ID: " +
        getText( xp, ctxNode, "cybs:requestID", nsNode ) );

    // Missing field or fields
    case 101:
      return( "\nThe following required field(s) are missing:\n" +
      enumerateValues( xp, ctxNode, "cybs:missingField", nsNode ) );
    // Invalid field or fields
    case 102:
      return( "\nThe following field(s) are invalid:\n" +
      enumerateValues( xp, ctxNode, "cybs:invalidField", nsNode ) );

    // Insufficient funds
    case 204:
      return( "\nInsufficient funds in the account. Please use a " +
              "different card or select another form of payment." );

    // Add additional reason codes here that you must handle specifically.
    default:
      // For all other reason codes (for example, unrecognized reason codes, or
      // codes that do not require special handling), return an empty string.
      return( "" );
  }
}

private static String enumerateValues(
  XPathAPI xp, Node ctxNode, String xpath, Node nsNode )
  throws TransformerException {
  try {
    StringBuffer sb = new StringBuffer();
    NodeList list =
      xp.selectNodeList( ctxNode, xpath + "/text()", nsNode );
    if (list != null) {
      for (int i = 0, len = list.getLength(); i < len; ++i) {
        sb.append( list.item( i ).getNodeValue() + "\n" );
      }
    }
    return( sb.toString() );
  }
}
```

Handling Decision Manager Reviews (CyberSource Advanced Merchants)

If you use CyberSource Decision Manager, you may also receive the `REVIEW` value in the **decision** field. `REVIEW` means that Decision Manager has marked the order for review based on how you configured the Decision Manager rules.

If you will be using Decision Manager, you have to determine how to handle the new `REVIEW` value. Ideally, you will update your order management system to recognize the `REVIEW` response and handle it according to your business rules. If you cannot update your system to handle the `REVIEW` response, CyberSource recommends that you choose one of these options:

- If you authorize and capture the credit card payment at the same time, treat the `REVIEW` response like a `REJECT` response. Rejecting any orders that are marked for review may be appropriate if your product is a software download or access to a Web site. If supported by your processor, you may also want to reverse the authorization.
- If you approve the order after reviewing it, convert the order status to `ACCEPT` in your order management system. You can request the credit card capture without requesting a new authorization.
- If you approve the order after reviewing it but cannot convert the order status to `ACCEPT` in your system, request a new authorization for the order. When processing this new authorization, you must disable Decision Manager. Otherwise the order will be marked for review again. For details about the API field that disables Decision Manager, see the *Decision Manager Developer Guide Using the Simple Order API* (PDF | HTML) or the *Decision Manager Developer Guide Using the SCMP Order API* (PDF | HTML).

Alternately, you can specify a custom business rule in Decision Manager so that authorizations originating from a particular internal IP address at your company are automatically accepted.

If supported by your processor, you may want to reverse the original authorization.

Handling System Errors

You must design your transaction management system to correctly handle CyberSource system errors, which occur when you successfully receive a reply, but the **decision** field is ERROR. For more information about the decision, see "Interpreting Replies," page 291. The error may indicate a valid CyberSource system error or a payment processor rejection because of invalid data.

Offline Transactions

CyberSource recommends that you resend the request two or three times only, waiting a longer period of time between each attempt. You should determine what is most appropriate for your business situation.

Example **After the first system error response, wait a short period of time, perhaps 30 seconds, before resending the request. If you receive the same error a second time, wait a longer period of time, perhaps 1 minute, before resending the request. If you receive the same error a third time, you may decide to try again after a longer period of time, perhaps 2 minutes.**

If you are still receiving a system error after several attempts, the error may be caused by a processor rejection instead of a CyberSource system error. In this case, CyberSource recommends one of these options:

- Find the transaction in the Business Center. After looking at the description of the error on the transaction details page, call your processor to determine if and why the transaction was rejected. If your processor is TSYS Acquiring Solutions, you may want to follow this option because this processor can return several system errors that only it can address.
- Contact CyberSource Customer Support to determine whether the error is caused by a CyberSource system issue.

Online Transactions

For online transactions, inform the customer that an error occurred and request that the customer attempts to resubmit the order.

Advanced Configuration Information

Using Alternate Server Properties

Use the serverURL and namespaceURI properties if CyberSource changes the convention used to specify the server URL and namespace URI, but has not updated the client yet. With these properties, you will be able to configure your existing client to use the new server and namespace conventions required by the CyberSource server.

For example, these are the server URLs and namespace URI for accessing the CyberSource services with the Simple Order API version 1.18:

- Test server URL:

  `https://ics2wstesta.ic3.com/commerce/1.x/transactionProcessor`

- Production server URL:

  `https://ics2wsa.ic3.com/commerce/1.x/transactionProcessor`

- Namespace URI:

  `urn:schemas-cybersource-com:transaction-data-1.18`.

If you view the above URLs in a web browser, a list of the supported API versions and the associated schema files are displayed.

Configuring for Multiple Merchant IDs

If you have multiple merchant IDs, or if you are a reseller handling multiple merchants, you can set different properties settings for different merchant IDs in the Properties object that you pass to runTransaction(). When using the samples provided in the client package, set the properties in `cybs.properties` file.

To specify the settings for a specific merchant, all the properties except merchantID can be prefixed with "`<merchantID>.`". The merchant ID is case sensitive. To enable logging only for `merchant123`, set the enableLog property to `true` for all requests that have `merchant123` as the merchant ID:

```
merchant123.enableLog=true
enableLog=false
```

The client disables logging for all other merchants.

Using System Properties

Although the properties in the Properties object passed to runTransaction() normally take precedence over the System properties, you can specify the settings that the client uses with the System properties. A system property will be used only if the Properties object passed to runTransaction() does not already include that property.

To use System properties for `merchant123`, prefix each system property with `cybs.`, for example:

```
java -Dcybs.enableLog=false -Dcybs.merchant123.enableLog=true
myApplication
```

Resolving Connection Issues

If you are using a Oracle Java SDK version earlier than 1.4.0 or an IBM Java SDK, you may exceptions when attempting to connect to external sites with HTTPS.

If you encounter the following exception message when testing the client, follow the procedure for your SDK:

```
java.net.MalformedURLException: unknown protocol: https
```

Oracle Java SDK version earlier than 1.4.0

This procedure is only a guideline. For the latest information, consult the Oracle JSSE documentation.

Step 1 Download the Oracle JSSE from http://java.sun.com/products/jsse/.

Step 2 Extract the following files from the Oracle JSSE package:

```
jcert.jar
jnet.jar
jsse.jar
```

Step 3 Copy the `jar` files into your Java installation's `jre/lib/ext` directory.

Step 4 Open `jre/lib/security/java.security` and locate the following line with the highest value for `N`:

```
security.provider.N=<some provider class name>
```

Step 5 Add the following line where `NN` is equal to `N` + 1:

```
security.provider.NN=com.sun.net.ssl.internal.ssl.Provider
```

Step 6 Save and close the file.

IBM Java SDK

This procedure is only a guideline. For the latest information, consult the IBMJSSE documentation.

Step 1 Download the IBMJSSE from IBM's web site or obtain it from your IBM development kit CDs.

Step 2 Extract the `ibmjsse.jar` file.

Step 3 Obtain the `ibmpkcs.jar` file.

The file should be included in the IBM development kit.

Step 4 Copy both `jar` files into your Java installation's `jre/lib/ext` directory.

Step 5 Open `jre/lib/security/java.security` and locate the following line with the highest value for `N`:

```
security.provider.N=<some provider class name>
```

Step 6 Add the following line where `NN` is equal to `N` + 1:

```
security.provider.NN=com.ibm.jsse.JSSEProvider
```

Step 7 Save and close the file.

Importing the Root CA Certificate

If you encounter this exception message when testing the client, you must perform the following steps to import the root CA certificate into `cacerts`:

```
javax.net.ssl.SSLException untrusted server cert chain
```

Step 1 At a command prompt, go to the main client directory where the `entrust_ssl_ca.cer` file is located.

Step 2 Type the following text without line breaks:

```
keytool -import -alias entrust_ssl_ca
                -keystore <JAVA_HOME>/jre/lib/security/cacerts
                -file entrust_ssl_ca.cer
```

where ***<JAVA_HOME>*** is the path to your Java installation.

Note that **`keytool`** is a utility included in the Java SDK.

Step 3 When prompted, enter the keystore password.

The default password is usually *changeit*. You have successfully imported the certificate.

CHAPTER

Perl Client

8

Important

- This chapter covers Linux and Windows platforms and uses the Linux convention of forward slashes when path names are listed.
- The Perl client for the Simple Order API is supported on 32-bit operating systems only.
- If you are building an application to sell to others, see Appendix A, "Using the Client Application Fields," on page 390. This appendix has a list of API fields you can use in your request that describe the application, its version, and its user. If you use these fields in your request, you can view their values in the Transaction Search Details window of the Business Center.

Using Perl in a Hosted Environment

If you are operating in a hosted environment (with an Internet Service Provider hosting your web store), then read this section.

To use the CyberSource Simple Order API client for Perl, you must install a Perl module from CyberSource. The CyberSource module ensures that your transactions are secure while being sent to CyberSource. If you use a hosted environment, you must check with your hosting provider (ISP) to make sure that they support the installation of the module.

If you are unable to find any documentation related to your hosting provider's support of new Perl modules, then contact them with the following statement:

> **CyberSource requires the installation of a CyberSource Perl module required for use by my e-commerce software. CyberSource ensures the safety and functionality of the module. Please let me know your policy for supporting this implementation.**

Note that it is also possible that other merchants who use your hosting provider may also use CyberSource, and so the hosting provider may have already installed the CyberSource Perl client. In that case, we suggest you verify with your hosting provider which version of the client they have installed and registered. If the client you want to use is newer, ask them to replace the module with the new one.

If you have any questions regarding the above information or installation of the client, please contact Customer Support.

Choosing Your API and Client

API Variation

With this client package, you can use either of these two variations of the Simple Order API:

- Name-value pairs, which are simpler to use than XML
- XML, which requires you to create and parse XML documents

The test that you run immediately after installing the client uses name-value pairs.

Client Versions

CyberSource regularly updates the Simple Order API to introduce new API fields and functionality. To identify the latest version of the server-side API for the CyberSource services, go to https://ics2wsa.ic3.com/commerce/1.x/transactionProcessor.

The Simple Order API Client for Perl also has a version, but it is not the same as the API version. The client version represents the version of the client-side code that you use to access the CyberSource services.

When configuring the client, you indicate which version of the API you want to use. When setting this parameter, do not use the current version of the client; use the current version of the API.

Sample Code

The client contains sample scripts and sample Perl pages that you can use to test the client.

Basic Perl Page Example

The example below shows the primary code required to send a Simple Order API request for credit card authorization and process the reply. The example uses name-value pairs. For a more complete example, see the sample scripts and sample store in the next sections. "Using Name-Value Pairs," page 321 shows you how to create the code.

```
use CyberSource::SOAPI;

# Load the configuration settings
%config = cybs_load_config( 'cybs.ini' );

# set up the request by creating a hash and adding fields to it
my %request;

# We want to do credit card authorization in this example
$request{'ccAuthService_run'} = 'true';
# Add required fields
$request{'merchantID'} = 'infodev';
$request{'merchantReferenceCode'} = 'MRC-14344';
$request{'billTo_firstName'} = 'Jane';
$request{'billTo_lastName'} = 'Smith';
$request{'billTo_street1'} = '1295 Charleston Road';
$request{'billTo_city'} = 'Mountain View';
$request{'billTo_state'} = 'CA';
$request{'billTo_postalCode'} = '94043';
$request{'billTo_country'} = 'US';
$request{'billTo_email'} = 'jsmith@example.com';
$request{'card_accountNumber'} = '4111111111111111';
$request{'card_expirationMonth'} = '12';
$request{'card_expirationYear'} = '2010';
$request{'purchaseTotals_currency'} = 'USD';
# This example has two items
$request{'item_0_unitPrice'} = '12.34';
$request{'item_1_unitPrice'} = '56.78';
# Add optional fields here according to your business needs

# Send request
my (%reply, $status);
$status = cybs_run_transaction(\%config, \%request, \%reply);

# Handle the reply. See "Handling the Return Status," page 324.
```

Sample Scripts

The client contains two sample scripts, one for using name-value pairs and one for using XML. See "Testing the Client," page 311 or see the README file for more information about using the `authCaptureSample.pl` script to test the client:

- Name-value pairs: See `authCaptureSample.pl` in *`<installation directory>`*`/samples/nvp`.
- XML: We suggest that you examine the name-value pair sample code listed above before implementing your code to process XML requests. For the XML sample code, see `authSample.pl` in *`<installation directory>`*`/samples/xml`. Also see the `auth.xml` XML document that the script uses.

Sample Store

The client download package also includes a sample store in the *`<installation directory>`*`/samples/store` directory.

Table 37 Perl Files in sampleStore Directory

File	Description
`checkout.pl`	Displays the contents of the shopping basket and prompts for address and payment information.
`checkout2.pl`	Authorizes the order and displays the result.
`store_footer.pl`	Footer used in the checkout pages.
`store_header.pl`	Header used in the checkout pages.
`storesample_util.pl`	File used by the other Perl files in the directory.

Note For Windows, the package also includes `checkout.plx` and `checkout2.plx`, which are similar to the `.pl` files. If you are using Perl CGI, use the `.pl` files. If you are using Perl ISAPI, use the `.plx` files.

To use the sample store:

Step 1 In the next step you will be copying the files listed in Table 37, "Perl Files in sampleStore Directory," on page 303 into the web server directory that contains your store's files. If you have files in that directory with the same names as the files in Table 37, page 303, make sure to back up your files first.

Step 2 Copy all of the files in the `<`*`installation directory`*`>/samples/store` directory into the web server directory you use to run your store.

Step 3 Modify the `cybs.ini` file in the `<`*`installation directory`*`>/samples/store` directory as appropriate. For more information, see "Configuring Client Settings," page 309.

Step 4 Open a web browser and type the following URL:

```
http://<your web server name or IP address>/<virtual directory if
applicable>/<checkout.pl or checkout.plx>
```

Installing and Testing the Client

Minimum System Requirements

- For Linux:

  Linux kernel 2.2, LibC6 on an Intel processor

  Minimum Perl version 5.006

  GNU GCC 3.1 or 3.1.1 compiler (with C++ enabled)

- For Windows:

  Windows XP, 2000, or later

  Either ActivePerl version 5.6 or 5.8

  The SDK supports UTF-8 encoding.

Important

Failure to configure your client API host to a unique, public IP address will cause inconsistent transaction results.

The client API request ID algorithm uses a combination of IP address and system time, along with other values. In some architectures this combination might not yield unique identifiers.

Transaction Security Keys

The first thing you must do is create your security key. The client uses the security key to add a digital signature to every request that you send. This signature helps ensure that no one else can use your CyberSource account to process orders. You specify the location of your key when you configure the client.

Important

You must generate two transaction security keys—one for the CyberSource production environment and one for the test environment. For information about generating and using security keys, see *Creating and Using Security Keys* (PDF | HTML).

The Simple Order API client for Perl package includes the `ca-bundle.crt`, a bundle of certificate files. The client expects to find the `ca-bundle.crt` file in the same directory as your security keys. If you move it elsewhere, use the sslCertFile configuration parameter to specify the file location. For more information, see the description of the parameter "sslCertFile," page 310.

Warning

You must protect your security key to ensure that your CyberSource account is not compromised.

Installing the Client

To install the client on Linux:

Step 1 Go to the client downloads page on the Support Center.

Step 2 Download the latest client package, and save the file in any directory.

Step 3 Unzip and untar the package.

This creates an installation directory called `simapi-perl-n.n.n`, where `n.n.n` is the client version.

Step 4 Make sure you are logged in as root (if you do not have root access, you should be working with your IT group or your Internet Service Provider to install the client).

Step 5 Generate the makefile by typing the following at a command prompt:

```
perl Makefile.PL
```

You may specify where you want the files to be copied by using the LIB parameter, for example:

```
perl Makefile.PL LIB=/home/user/perllib
```

Step 6 Enter these commands:

```
make
make test
make install
```

Important

When building the Perl extension `SOAPI.so`, MakeMaker uses the `lib` subdirectory of the client as the `LD_RUN_PATH`, which makes `SOAPI.so` dependent on the full `lib` path. Therefore, you must choose one of these options:

- Keep the client's `lib` subdirectory and its files in their current location.
- Include your Perl's architecture-specific directory (for example: `<perl root dir>/lib/site_perl/5.8.8/i686-linux-thread-multi`) in your `LD_LIBRARY_PATH`. The `make install` step copies the shared libraries to that location.

The client is installed on your system.

Step 7 Configure the client. See "Configuring Client Settings," page 309 below.

Step 8 Test the client. See "Testing the Client," page 311.

Important

If you are upgrading from a pre-5.0.0 version of the CyberSource client, you must update your code to use the package `CyberSource::SOAPI` instead of `cybs`. Also, you can now omit the package prefix (`cybs::`) when using the constants defined by the client.

The client is installed and tested. You are ready to create your own code for requesting CyberSource services. Finish reading this section, and then move on to either "Using Name-Value Pairs," page 321 if you plan to use name-value pairs, or "Using XML," page 331 if you plan to use XML.

To install the client on Windows:

Step 1 Go to the client downloads page on the Support Center.

Step 2 Download the latest client package. You can save the file in any directory.

Step 3 Unzip the package into a directory of your choice.

This creates an installation directory called `simapi-perl-n.n.n`, where `n.n.n` is the client version.

Step 4 Change to the `simapi-perl-n.n.n` directory.

Step 5 If you have a previous version of the CyberSource Simple Order API client, uninstall it with PPM3:

```
ppm3 uninstall CyberSource-SOAPI
```

Step 6 Install the ppd file with PPM3 by typing this command at a prompt:

```
ppm3 install CyberSource-SOAPI.ppd
```

The client is now installed on your system.

Step 7 Configure the client. See "Configuring Client Settings," page 309 below.

Step 8 Test the client. See "Testing the Client," page 311.

The client is installed and tested. You are ready to create the code for requesting CyberSource services. Depending on your implementation, continue either with "Using Name-Value Pairs," page 321 or "Using XML," page 331.

Configuring Client Settings

To run the sample scripts included in the client package, you must set the configuration parameters in the `cybs.ini` file, which is located in the installation directory. You can also use this file when running transactions in a production environment (see the function descriptions in "Perl API for the Client," page 314). The following table describes the parameters that you can set. Note that the default `cybs.ini` file that comes with the client package does not include all of the parameters listed in the table. It includes only the ones required to run the sample scripts.

If you have multiple merchant IDs, or if you are a reseller handling multiple merchants, you can use different configuration settings depending on the merchant ID. See "Configuring for Multiple Merchant IDs," page 344 for more information.

Table 38 Configuration Settings

Setting	Description
merchantID	Merchant ID. This client uses this value if you do not specify a merchant ID in the request itself.
keysDirectory	Location of the merchant's security key. The client includes a `keys` directory that you can use. **Note** We recommend that you store your key locally for faster request processing.
sendToProduction	Flag that indicates whether the transactions for this merchant should be sent to the production server. Use one of these values: ■ `false`: Do not send to the production server; send to the test server (default setting). ■ `true`: Send to the production server.
targetAPIVersion	Version of the Simple Order API to use. For example, `1.18`. Do not set this property to the current version of the client; set it to an available API version. See "Client Versions," page 301 for more information. **Note** Go to https://ics2wsa.ic3.com/commerce/1.x/transactionProcessor to see a current list of the available versions. See the *Simple Order API Release Notes*, for information about what has changed in each version.
keyFilename	Name of the security key filename for the merchant in the format `<`***`security_key_filename`***`>.p12`.
serverURL	Alternative server URL to use. See "Using Alternate Server Configuration Settings," page 343 for more information. Give the complete URL because it will be used exactly as you specify here.
namespaceURI	Alternative namespace URI to use. See "Using Alternate Server Configuration Settings," page 343 for more information. Give the complete namespace URI because it will be used exactly as you specify here.

Table 38 Configuration Settings (Continued)

Setting	Description
enableLog	Flag directing the client to log transactions and errors. Use one of these values: ■ `false`: Do not enable logging (default setting). ■ `true`: Enable logging. **Important** Logging can cause very large log files to accumulate. Therefore, CyberSource recommends that you use logging only when troubleshooting problems. To comply with all Payment Card Industry (PCI) and Payment Application (PA) Data Security Standards regarding the storage of credit card and card verification number data, the logs that are generated contain only masked credit card and card verification number data (CVV, CVC2, CVV2, CID, CVN). Follow these guidelines: ■ Use debugging temporarily for diagnostic purposes only. ■ If possible, use debugging only with test credit card numbers. ■ Never store clear text card verification numbers. ■ Delete the log files as soon as you no longer need them. ■ Never send email to CyberSource containing personal and account information, such as customers' names, addresses, card or check account numbers, and card verification numbers. For more information about PCI and PABP requirements, see www.visa.com/cisp.
logDirectory	Directory to which to write the log file. Note that the client will not create this directory for you; you must specify an existing directory.The client includes a `logs` directory that you can use.
logFilename	Log file name. The client uses `cybs.log` by default.
logMaximumSize	Maximum size in megabytes for the log file. The default value is `"10"`. When the log file reaches the specified size, it is archived into `cybs.log.`***<yyyymmdd***`T`***hhmmssxxx>*** and a new log file is started. The ***xxx*** indicates milliseconds.
sslCertFile	The location of the bundled file of CA Root Certificates (`ca-bundle.crt`) which is included in the client download package. The client automatically looks for the file in the directory where your security key is stored (specified by keysDirectory). If you move the file so it does not reside in keysDirectory, use this configuration setting to specify the full path to the file, including the file name.
timeout	Length of timeout in seconds. The default is 110.

Table 38 Configuration Settings (Continued)

Setting	Description
proxyServer	Proxy server to use. Allowable formats include: ▪ *<http://>server<:port>* ▪ *<http://>IP address<:port>* The `http://` and `port` are optional. **Note** The default port is 1080. If your proxy server is listening on another port, you must specify a port number.
proxyUsername	User name used to authenticate against the proxy server if required. If the proxy server requires the domain name during authentication, add the domain name and a backslash: ***<domain>**<username>***
proxyPassword	Password used to authenticate against the proxy server if required.

Testing the Client

After the client is installed and configured, immediately test it to ensure the installation is successful.

To test the client:

Step 1 Go to the `<installation directory>/samples/nvp` directory.

Step 2 Run the test `authCaptureSample.pl` script by typing:

```
perl authCaptureSample.pl
```

Test results are displayed in the window.

- If the test is successful, a decision of ACCEPT appears for both the credit card authorization and the follow-on capture.
- If the test is not successful, a different decision value or an error message appears.

To troubleshoot a client test failure:

Step 1 Verify that your `cybs.ini` settings are correct.

Step 2 Run the test again.

Step 3 If the test still fails, look at the error message and find the return status value (a numeric value from -1 to 8).

Step 4 See the descriptions of the status values in "Possible Return Status Values," page 316, and follow any instructions given there for the error you received.

Step 5 Run the test again.

Step 6 If the test still fails, contact Customer Support.

To run the XML sample:

Step 1 Go to the `<installation directory>/samples/xml` directory.

Step 2 Run the test `authSample.pl` script by typing:

```
perl authSample.pl
```

The results of the test are displayed in the window.

- If the test is successful, a decision of ACCEPT appears for both the credit card authorization and the follow-on capture.
- If the test is not successful, a different decision value or an error message appears. See "To troubleshoot a client test failure:," page 312 for information about troubleshooting the error.

Going Live

When you complete all of your system testing and are ready to accept real transactions from your customers, your deployment is ready to *go live*.

Note

After your deployment goes live, use real card numbers and other data to test every card type you support. Because these are real transactions in which you are buying from yourself, use small monetary amounts to do the tests. Process an authorization, then capture the authorization, and later refund the money. Use your bank statements to verify that money is deposited into and withdrawn from your merchant bank account as expected. If you have more than one CyberSource merchant ID, test each one separately.

CyberSource Essentials Merchants

If you use CyberSource Essentials services, you can use the Business Center site to go live. For a description of the process of going live, see the “Steps for Getting Started” section in *Getting Started with CyberSource Essentials*.

Important

Configure your client so that it can send transactions to the production server and not the test server. See the description of the sendToProduction property in Table 38, "Configuration Settings," on page 309.

CyberSource Advanced Merchants

If you use CyberSource Advanced services, see the “Steps for Getting Started” chapter in *Getting Started with CyberSource Advanced* for information about going live.

When you go live, your CyberSource account is updated so that you can send transactions to the CyberSource production server. If you have not already done so, you must provide your banking information to CyberSource so that your processor can deposit funds to your merchant bank account.

After CyberSource has confirmed that your account is live, make sure that you update your system so that it can send requests to the production server (`ics2wsa.ic3.com`) using your security key for the production environment. The test server (`ics2wstesta.ic3.com`) cannot be used for real transactions. For more information about sending transactions to the production server, see the description of the property "sendToProduction," page 309.

Updating the Client to Use a Later API Version

CyberSource periodically updates the Simple Order API (previously called the Web Services API). You can update your existing client to work with the new API version. Go to https://ics2wsa.ic3.com/commerce/1.x/transactionProcessor for a list of the available API versions.

To update the client to use a later API version, update the value for the `targetAPIVersion` configuration parameter. For example, to use the 1.18 version of the API, set the property to `1.18`.

Perl API for the Client

Important If you use a constant name as a key to a hash and do not fully qualify the constant name, you must prefix the constant name with a plus sign (+), for example: `$request{+CYBS_SK_XML_DOCUMENT}`.

Summary of Functions

The client includes these functions:

- cybs_load_config()
- cybs_run_transaction()

cybs_load_config()

Table 39 cybs_load_config()

Syntax	`hash cybs_load_config( string filename )`
Description	Loads the configuration settings from a file.
Returns	A hash containing the configuration settings.
Parameters	`filename`: Name of the configuration file.

cybs_run_transaction()

Table 40 cybs_run_transaction()

Syntax	`int cybs_run_transaction( hash-pointer pconfig, hash-pointer prequest, hash-pointer preply )`	
Description	Sends the request to the CyberSource server and receives the reply.	
Returns	A value that indicates the status of the request.	
Parameters	`config`: Configuration hash to use.	
	`request`: Hash containing one of these:	- The individual name-value pairs in the request (for name-value pair users) - A single key called `CYBS_SK_XML_DOCUMENT` whose value is the XML document representing the request (for XML users) To use fully qualified constant names, see the note on page 17.

	`reply`:
	Hash containing one of these:
		Note You must create this hash before you call cybs_run_transaction().
		To use fully qualified constant names, see the note on page 17.
		- The individual name-value pairs in the reply (for name-value pair users)
		- A single key called `CYBS_SK_XML_DOCUMENT` whose value is the XML document representing the reply (for XML users)
		- A combination of the following keys and their values:
		`CYBS_SK_ERROR_INFO`
		`CYBS_SK_RAW_REPLY`
		`CYBS_SK_FAULT_DOCUMENT`
		`CYBS_SK_FAULT_CODE`
		`CYBS_SK_FAULT_STRING`
		`CYBS_SK_FAULT_REQUEST_ID`
		See below for descriptions of these keys.

Reply Key Descriptions

Note

If you use a constant name as a key to a hash and do not fully qualify the constant name, you must prefix the constant name with a plus sign (+), for example: `$request{+CYBS_SK_XML_DOCUMENT}`.

- `CYBS_SK_ERROR_INFO`: Information about the error that occurred
- `CYBS_SK_RAW_REPLY`: The server's raw reply
- `CYBS_SK_FAULT_DOCUMENT`: The entire, unparsed fault document
- `CYBS_SK_FAULT_CODE`: The fault code, which indicates where the fault originated
- `CYBS_SK_FAULT_STRING`: The fault string, which describes the fault.
- `CYBS_SK_FAULT_REQUEST_ID`: The request ID for the request.

Possible Return Status Values

The cybs_run_transaction() function returns a status indicating the result of the request. Table 41, page 316 describes the possible status values, including whether the error is critical. If an error occurs after the request has been sent to the server, but the client cannot determine whether the transaction was successful, then the error is considered critical. If a critical error happens, the transaction may be complete in the CyberSource system but not complete in your order system. The descriptions below indicate how to handle critical errors.

Note

- The sample scripts display a numeric value for the return status, which is listed in the first column.
- If you use a constant name as a key to a hash and do not fully qualify the constant name, you must prefix the constant name with a plus sign (+), for example: `$request{+CYBS_SK_XML_DOCUMENT}`.

The numeric value in the first column applies to sample scripts.

Table 41 Possible Status Values

Numeric Value	Value	Description
0	`CYBS_S_OK`	**Critical:** No **Result:** The client successfully received a reply. For name-value pair users, the `$reply` hash has the reply name-value pairs for the services that you requested. For XML users, the `$reply` hash contains the response in XML format. **Manual action to take:** None

Table 41 Possible Status Values (Continued)

Numeric Value	Value	Description
-1	CYBS_S_PERL_ PARAM_ERROR	**Critical:** No **Result:** The request was not sent because a problem occurred with one or more of the parameters passed to the cybs_run_transaction() function. **Manual action to take:** Make sure the parameter values are correct.
1	CYBS_S_PRE_SEND_ ERROR	**Critical:** No **Result:** An error occurred before the request could be sent. This usually indicates a configuration problem with the client. **Error information to read:** `$reply{+CYBS_SK_ERROR_INFO}` **Manual action to take:** Fix the problem described in the error information.
2	CYBS_S_SEND_ERROR	**Critical:** No **Result:** An error occurred while sending the request. **Error information to read:** `$reply{+CYBS_SK_ERROR_INFO}` **Manual action to take:** None **Note** A typical send error that you might receive when testing occurs if the `ca-bundle.crt` file is not located in the same directory as your security key. To correct the problem, see the description of the `sslCertFile` configuration parameter in Table 38, "Configuration Settings," on page 309.

3	CYBS_S_RECEIVE_
	ERROR
		Critical: Yes
		Result: An error occurred while waiting for or retrieving the reply.
		Error information to read:
		`$reply{+CYBS_SK_ERROR_INFO}`
		`$reply{+CYBS_SK_RAW_REPLY}`
		Manual action to take: Check the Transaction Search screens on the Business Center to verify that the request was processed, and if so, whether it succeeded. Update your transaction database appropriately.

Table 41 Possible Status Values (Continued)

Numeric Value	Value	Description

4	CYBS_S_POST_
	RECEIVE_ERROR
		Critical: Yes
		Result: The client received a reply or a fault, but an error occurred while processing it.
		Error information to read:
		`$reply{+CYBS_SK_ERROR_INFO}`
		`$reply{+CYBS_SK_RAW_REPLY}`
		Manual action to take: Examine the value of `$reply{CYBS_SK_RAW_REPLY}`. If you cannot determine the status of the request, check the Transaction Search screens on the Business Center to verify that the request was processed, and if so, whether it succeeded. Update your transaction database appropriately.

5	CYBS_S_CRITICAL_
	SERVER_FAULT
		Critical: Yes
		Result: The server returned a fault with `$reply{+CYBS_SK_FAULT_CODE}` set to CriticalServerError.
		Error information to read:
		`$reply{+CYBS_SK_ERROR_INFO}`
		`$reply{+CYBS_SK_FAULT_CODE}`
		`$reply{+CYBS_SK_FAULT_STRING}`
		`$reply{+CYBS_SK_FAULT_DOCUMENT}`
		`$reply{+CYBS_SK_FAULT_REQUEST_ID}`
		Manual action to take: Check the Transaction Search screens on the Business Center to verify that the request succeeded. When searching for the request, use the request ID provided by `$reply{CYBS_SK_FAULT_REQUEST_ID}`.

6	CYBS_S_SERVER_
	FAULT
		Critical: No
		Result: The server returned a fault with `$reply{+CYBS_SK_FAULT_CODE}` set to ServerError, indicating a problem with the CyberSource server.
		Error information to read:
		`$reply{+CYBS_SK_ERROR_INFO}`
		`$reply{+CYBS_SK_FAULT_CODE}`
		`$reply{+CYBS_SK_FAULT_STRING}`
		`$reply{+CYBS_SK_FAULT_DOCUMENT}`
		Manual action to take: None

Table 41 Possible Status Values (Continued)

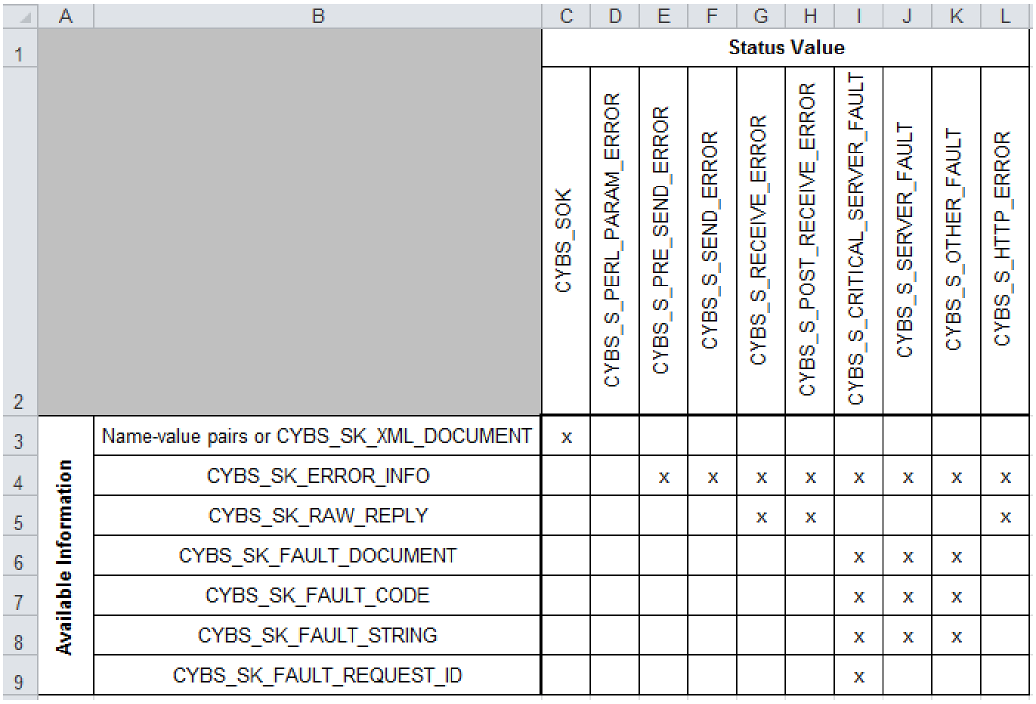
Numeric Value	Value	Description

7	CYBS_S_OTHER_FAULT	**Critical:** No
		Result: The server returned a fault with `$reply{+CYBS_SK_FAULT_CODE}` set to a value other than ServerError or CriticalServerError. Indicates a possible problem with merchant status or the security key or that the message was tampered with after it was signed and before it reached the CyberSource server.
		Error information to read:
		`$reply{+CYBS_SK_ERROR_INFO}`
		`$reply{+CYBS_SK_FAULT_CODE}`
		`$reply{+CYBS_SK_FAULT_STRING}`
		`$reply{+CYBS_SK_FAULT_DOCUMENT}`
		Manual action to take: Examine the value of the `$reply{CYBS_SK_FAULT_STRING}` and fix the problem. You might need to generate a new security key or to contact Customer Support if problems exist with your merchant status.
		Note A typical error occurs if your merchant ID is configured for test mode, but you send transactions to the production server. To correct the problem, see the `sendToProduction` configuration parameter in Table 38, "Configuration Settings," on page 309.

8	`CYBS_S_HTTP_ERROR`	**Critical:** No
		Result: The server returned an HTTP status code other than 200 (OK) or 504 (gateway timeout). Note that if a 504 gateway timeout occurs, the status=`3`.
		Error information to read:
		`$reply{+CYBS_SK_ERROR_INFO}`
		`$reply{+CYBS_SK_RAW_REPLY}`
		Value of varReply: `CYBS_SK_RAW_REPLY` contains the HTTP response body, or if none was returned, the literal `"(no response available)"`.
		Manual action to take: None.

Table 42 summarizes the reply information that you receive for each status value.

Table 42 Reply Information Available for Each Status Value



	A	B	C	D	E	F	G	H	I	J	K	L
1			**Status Value**									
2			CYBS_SOK	CYBS_S_PERL_PARAM_ERROR	CYBS_S_PRE_SEND_ERROR	CYBS_S_SEND_ERROR	CYBS_S_RECEIVE_ERROR	CYBS_S_POST_RECEIVE_ERROR	CYBS_S_CRITICAL_SERVER_FAULT	CYBS_S_SERVER_FAULT	CYBS_S_OTHER_FAULT	CYBS_S_HTTP_ERROR
3	**Available Information**	Name-value pairs or CYBS_SK_XML_DOCUMENT	x									
4		CYBS_SK_ERROR_INFO			x	x	x	x	x	x	x	x
5		CYBS_SK_RAW_REPLY					x	x				x
6		CYBS_SK_FAULT_DOCUMENT							x	x	x	
7		CYBS_SK_FAULT_CODE							x	x	x	
8		CYBS_SK_FAULT_STRING							x	x	x	
9		CYBS_SK_FAULT_REQUEST_ID							x			

Using Name-Value Pairs

This section explains how to use the client to request CyberSource services by using name-value pairs.

Requesting CyberSource Services

To request CyberSource services, write code that:

- Collects information for the CyberSource services that you will use
- Assembles the order information into requests
- Sends the requests to the CyberSource server
- Processes the reply information

Important The CyberSource servers do not support persistent HTTP connections.

The instructions in this guide explain how to use Perl to request CyberSource services. For a list of API fields to use in your requests, see "Related Documents," page 21.

Creating and Sending Requests

Important The code in this section's example is incomplete. For a complete sample program, see the `authCaptureSample.pl` file in the *`<installation directory>`*`/samples/nvp` directory, or see the sample web pages.

If you use a constant name as a key to a hash and do not fully qualify the constant name, you must prefix the constant name with a plus sign (+), for example: `$request{+CYBS_SK_XML_DOCUMENT}`.

To use any CyberSource service, you must create and send a request that includes the required information for that service.

The example that is developed in the following sections shows basic code for requesting CyberSource services. In this example, Jane Smith is buying an item for $29.95.

Adding the Use Statement

First add the use statement for the `CyberSource::SOAPI` module:

```
use CyberSource::SOAPI;
```

Loading the Configuration Settings

Next load the configuration settings from a file:

```
%config = cybs_load_config( 'cybs.ini' );
```

You could instead create a hash and add each configuration setting separately. You could also use a combination of the two methods: You could read the settings from a file and then add new settings dynamically with the hash to override the settings read from the file.

Creating an Empty Request Hash

You next create a hash to hold the request fields:

```
my %request;
```

Adding the Merchant ID

You next add the CyberSource merchant ID to the request. You can let the CyberSource Perl client automatically retrieve the merchant ID from the `%config` hash, or you can set it directly in the `%request` hash (see below). The `%request` hash value overrides the `%config` hash value.

```
$request{'merchantID'} = 'infodev';
```

Adding Services to the Request Hash

You next indicate the service you want to use by adding the field to the request. For example, to request a credit card authorization:

```
$request{'ccAuthService_run'} = 'true';
```

Requesting a Sale

You can request multiple services by adding additional fields to the request. For example, if you fulfill the order immediately, you can request credit card authorization and capture together (referred to as a “sale”):

```
$request{'ccAuthService_run'} = 'true';
$request{'ccCaptureService_run'} = 'true';
```

Adding Service-Specific Fields to the Request Hash

You next add the fields that are used by the services that you are requesting. If you request multiple services and they share common fields, you must add the field once only.

```
$request{'merchantReferenceCode'} = '3009AF229L7W';
$request{'billTo_firstName'} = 'Jane';
$request{'billTo_lastName'} = 'Smith';
$request{'card_accountNumber'} = '4111111111111111';
$request{'item_0_unitPrice'} = '29.95';
```

The example above shows only a partial list of the fields you must send. See "Related Documents," page 21 for information about other guides that list the API fields for the services you are requesting.

Sending the Request

You next create the hash that will hold the reply and send the request:

```
my (%reply, $status);
$status = cybs_run_transaction( \%config, \%request, \%reply );
```

Interpreting Replies

Handling the Return Status

The `$status` value is the handle returned by the cybs_run_transaction() method. (See the following example.) The `$status` indicates whether the CyberSource server received the request, the client received the reply, or there were any errors or faults during transmission. See "Possible Return Status Values," page 316 for descriptions of each status value. For a different example, see the `authCaptureSample.pl` file in the *<installation directory>*`/sample` directory.

```
if ($status == CYBS_S_OK) {
  # Read the value of the "decision" in the %reply hash.
  $decision = $reply{'decision'};
  # If decision=ACCEPT, indicate to the customer that the request was successful.
  # If decision=REJECT, indicate to the customer that the order was not approved.
  # If decision=ERROR, indicate to the customer an error occurred and to try again
  # later.

  # Now get reason code results:
  # $strContent = getReplyContent(\%reply);
  # See "Processing the Reason Codes," page 326 for how to process the reasonCode
  # from the reply.
  # Note that getReplyContent() is included in this document to help you understand
  # how to process reason codes, but it is not included as part of the sample scripts
  # or sample web pages.
} else {
  handleError( $status, \%request, \%reply );
}
#---------------------
sub handleError
#---------------------
{
  # handleError shows how to handle the different errors that can occur.
  # To use fully qualified constant names, see the note in "Creating and Sending
Requests," page 321.
  my ($nStatus, $pRequest, $pReply) = @_;

  # There was a problem with the parameters passed to cybs_run_transaction()
  if ( $nStatus == CYBS_S_PERL_PARAM_ERROR ) {
    # Non-critical error.
    # Tell customer the order cannot be completed and to try again later.
    # Notify appropriate internal resources of the error.
  }
  # An error occurred before the request could be sent.
  elsif ( $nStatus == CYBS_S_PRE_SEND_ERROR ) {
    # Non-critical error.
    # Tell customer the order cannot be completed and to try again later.
    # Notify appropriate internal resources of the error.
  }

  # An error occurred while sending the request.
  elsif ( $nStatus == CYBS_S_SEND_ERROR ) {
    # Non-critical error.
    # Tell customer the order cannot be completed and to try again later.
  }
  # An error occurred while waiting for or retrieving the reply.
  elsif ( $nStatus == CYBS_S_RECEIVE_ERROR ) {
    # Critial error.
    # Tell customer the order cannot be completed and to try again later.
    # Notify appropriate internal resources of the error.
    # See the sample code for more information about handling critical errors.
  }

  # An error occurred after receiving and during processing of the reply.
  elsif ( $nStatus == CYBS_S_POST_RECEIVE_ERROR ) {
    # Critical error.
    # Tell customer the order cannot be completed and to try again later.
    # Look at CYBS_SK_RAW_REPLY in $reply for the raw reply.
    # Notify appropriate internal resources of the error.
    # See the sample code for more information about handling critical errors.
  }

  # CriticalServerError fault
  elsif ( $nStatus == CYBS_S_CRITICAL_SERVER_FAULT ) {
    # Critial error.
    # Tell customer the order cannot be completed and to try again later.
    # Read the various fault details from the $reply.
    # Notify appropriate internal resources of the fault.
    # See the sample code for more information about reading fault details and
    # handling a critical error.
  }

  # ServerError fault
  elsif ( $nStatus == CYBS_S_SERVER_FAULT ) {
    # Non-critical error.
    # Tell customer the order cannot be completed and to try again later.
    # Read the various fault details from the $reply.
    # See the sample code for information about reading fault details.
  }
  # Other fault
  elsif ( $nStatus == CYBS_S_OTHER_FAULT ) {
    # Non-critical error.
    # Tell customer the order cannot be completed and to try again later.
    # Read the various fault details from the $reply.
    # Notify appropriate internal resources of the fault.
    # See the sample code for information about reading fault details.
  }

  # HTTP error
  elsif ( $nStatus == CYBS_S_HTTP_ERROR ) {
    # Non-critical error.
    # Tell customer the order cannot be completed and to try again later.
    # Look at CYBS_SK_RAW_REPLY in $reply for the raw reply.
    break;
  }
}
```

Processing the Reason Codes

After the CyberSource server processes your request, it sends a reply message that contains information about the services you requested. (See following example.) You receive different fields depending on the services you request and the outcome of each service.

To use the reply information, you must integrate it into your system and any other system that uses that data. For example, you can store the reply information in a database and send it to other back office applications.

You must write an error handler to process the reply information that you receive from CyberSource. Do not show the reply information directly to customers. Instead, present an appropriate response that tells customers the result.

Important

Because CyberSource may add reply fields and reason codes at any time, you should parse the reply data according to the names of the fields instead of their order in the reply.

The most important reply fields to evaluate are the following:

- **decision**: A one-word description of the results of your request. The decision is one of the following:
  - `ACCEPT` if the request succeeded
  - `REJECT` if one or more of the services in the request was declined
  - `REVIEW` if you are CyberSource Advanced merchant using CyberSource Decision Manager and it flags the order for review. See "Handling Decision Manager Reviews," page 328 for more information.

  - `ERROR` if there was a system error. See "Retrying When System Errors Occur," page 330 for more information.

- **reasonCode**: A numeric code that provides more specific information about the results of your request.

You also receive a reason code for each service in your request. You can use these reason codes to determine whether a specific service succeeded or failed. If a service fails, other services in your request may not run. For example, if you request a credit card authorization and capture, and the authorization fails, the capture does not run. The reason codes for each service are described in the *Credit Card Services User Guide* for CyberSource Essentials merchants or in the service developer guide for CyberSource Advanced merchants.

Important CyberSource reserves the right to add new reason codes at any time. If your error handler receives a reason code that it does not recognize, it should use the decision to interpret the reply.

```
# Note that getReplyContent is included in this document to help you understand how to
# process reason codes, but it is not included as part of the sample scripts or
# sample web pages.
#----------------
sub getReplyContent( $pReply )
#----------------
{
  my ($pReply) = @_;
  $reasonCode = $$pReply{'reasonCode'};
  # Success
  if ($reasonCode == '100'){
    return( sprintf(
      "Request ID: %s\nAuthorizedAmount: %s\nAuthorization Code: %s,
      $$pReply{'requestID'}, $$pReply{'ccAuthReply_amount'},
      $$pReply{'ccAuthReply_authorizationCode'} ) );
  }

  # Insufficient funds
  elsif ($reasonCode == '204'){
    return( sprintf(
    "Insufficient funds in account. Please use a different card or select
       another form of payment." ) );
  }

    # add other reason codes here that you must handle specifically

  else {
    # For all other reason codes, return NULL, in which case, you should display a
    # generic message appropriate to the decision value you received.
    return( NULL );
  }
}
```

Handling Decision Manager Reviews

If you use CyberSource Decision Manager, you may also receive the `REVIEW` value in the **decision** field. `REVIEW` means that Decision Manager has marked the order for review based on how you configured the Decision Manager rules.

If you will be using Decision Manager, you have to determine how to handle the new `REVIEW` value. Ideally, you will update your order management system to recognize the `REVIEW` response and handle it according to your business rules. If you cannot update your system to handle the `REVIEW` response, CyberSource recommends that you choose one of these options:

- If you authorize and capture the credit card payment at the same time, treat the `REVIEW` response like a `REJECT` response. Rejecting any orders that are marked for review may be appropriate if your product is a software download or access to a Web site. If supported by your processor, you may also want to reverse the authorization.

- If you approve the order after reviewing it, convert the order status to `ACCEPT` in your order management system. You can request the credit card capture without requesting a new authorization.

- If you approve the order after reviewing it but cannot convert the order status to `ACCEPT` in your system, request a new authorization for the order. When processing this new authorization, you must disable Decision Manager. Otherwise the order will be marked for review again. For details about the API field that disables Decision Manager, see the *Decision Manager Developer Guide Using the Simple Order API* (PDF | HTML) or the *Decision Manager Developer Guide Using the SCMP Order API* (PDF | HTML).

Alternately, you can specify a custom business rule in Decision Manager so that authorizations originating from a particular internal IP address at your company are automatically accepted.

If supported by your processor, you may want to reverse the original authorization.

Requesting Multiple Services

When you request multiple services in one request, CyberSource processes the services in a specific order. If a service fails, CyberSource does not process the subsequent services in the request.

For example, in the case of a sale (a credit card authorization and a capture requested together), if the authorization service fails, CyberSource will not process the capture service. The reply you receive only includes reply fields for the authorization.

This following additional example applies **to CyberSource Advanced merchants only.**

Many CyberSource services include “ignore” fields that tell CyberSource to ignore the result from the first service when deciding whether to run the subsequent services. In the case of the sale, even though the issuing bank gives you an authorization code, CyberSource might decline the authorization based on the AVS or card verification results. Depending on your business needs, you might choose to capture these types of declined authorizations anyway. You can set the **businessRules_ignoreAVSResult** field to “true” in your combined authorization and capture request:

```
$request{'businessRules_ignoreAVSResult'} = 'true';
```

This tells CyberSource to continue processing the capture even if the AVS result causes CyberSource to decline the authorization. In this case you would then get reply fields for both the authorization and the capture in your reply.

Note

You are charged only for the services that CyberSource performs.

Retrying When System Errors Occur

You must design your transaction management system to include a way to correctly handle CyberSource system errors. Depending on which payment processor is handling the transaction, the error may indicate a valid CyberSource system error, or it may indicate a processor rejection because of some type of invalid data. In either case, CyberSource recommends that you do not design your system to retry sending a transaction many times in the case of a system error.

Instead, CyberSource recommends that you retry sending the request only two or three times with successively longer periods of time between each retry. For example, after the first system error response, wait 30 seconds and then retry sending the request. If you receive the same error a second time, wait one minute before you send the request again. Depending on the situation, you may decide you can retry sending the request after a longer time period. Determine what is most appropriate for your business situation.

If after several retry attempts you are still receiving a system error, it is possible that the error is actually being caused by a processor rejection and not a CyberSource system error. In that case, we suggest that you either:

- Search for the transaction in the Business Center, look at the description of the error on the Transaction Detail page, and call your processor to determine if and why they are rejecting the transaction.

- Contact CyberSource Customer Support to confirm whether your error is truly caused by a CyberSource system issue.

If TSYS Acquiring Solutions is your processor, you may want to follow the first suggestion as there are several common TSYS Acquiring Solutions processor responses that are returned to you as system errors and that only TSYS Acquiring Solutions can address.

Using XML

This section describes how to request CyberSource services by using XML.

Requesting CyberSource Services

To request CyberSource services, write code that:

- Collects information for the CyberSource services that you will use
- Assembles the order information into requests
- Sends the requests to the CyberSource server

Important

You are charged only for the services that CyberSource performs.

- Processes the reply information

The instructions in this section explain how to use Perl to request CyberSource services. For a list of API fields to use in your requests, see "Related Documents," page 21.

Sample Code

We suggest that you examine the name-value pair sample code provided in `authCaptureSample.pl` before implementing your code to process XML requests. The sample will give you a basic understanding of how to request CyberSource services. The sample code file is located in the *`<installation directory>`*`/samples/nvp` directory.

After examining that sample code, read this section to understand how to create code to process XML requests. Note that the code in this section's example is incomplete. For a complete sample program, see the `authSample.pl` file in the *`<installation directory>`*`/samples/xml` directory.

Creating a Request Document

Important

If you use a constant name as a key to a hash and do not fully qualify the constant name, you must prefix the constant name with a plus sign (+), for example: `$request{+CYBS_SK_XML_DOCUMENT}`.

The client allows you to create an XML request document using any application, then send the request to CyberSource. For example, if you have a customer relationship management (CRM) system that uses XML to communicate with other systems, you can use the CRM system to generate request documents.

The request document must validate against the XML schema for CyberSource transactions. To view the schema, go to https://ics2wsa.ic3.com/commerce/1.x/transactionProcessor and look at the `xsd` file for the version of the Simple Order API you are using.

Important

Make sure that the elements in your document appear in the correct order. If they do not, your document will not validate, and your request will fail.

The following example shows a basic XML document for requesting CyberSource services. In this example, Jane Smith is buying an item for $29.95.

The XML document in this example is incomplete. For a complete example, see the `auth.xml` document in the `samples/xml` directory.

Creating an Empty Request

Add the XML declaration and the document's root element:

```
<?xml version="1.0" encoding="utf-8"?>
<requestMessage xmlns="urn:schemas-cybersource-com:transaction-data-
1.18">
</requestMessage>
```

When you construct a request, you must indicate the correct namespace for the elements, and the namespace must use the same API version that you specify in the configuration settings file. For example, if `targetAPIVersion=1.18` in the `cybs.ini` file, the namespace must be `urn:schemas-cybersource-com:transaction-data-1.18`.

Note

The XML document that you receive in the reply always uses a prefix of `c:` (for example, `xmlns:c="urn:schemas-cybersource-com:transaction-data-1.18"`). Make sure you use an XML parser that supports namespaces.

Adding the Merchant ID

You next add the CyberSource merchant ID to the request.

Note If you specify a merchant ID in the XML document, it overrides the merchant ID you specify in the configuration settings file.

```
<requestMessage xmlns="urn:schemas-cybersource-com:transaction-data-
1.18">
   <merchantID>infodev</merchantID>
</requestMessage>
```

Adding Services to the Request

You next indicate the service that you want to use by creating an element for that service in the request, then setting the element's `run` attribute to `true`. For example, to request a credit card authorization:

```
<requestMessage xmlns="urn:schemas-cybersource-com:transaction-data-
1.18">
  <merchantID>infodev</merchantID>
  <ccAuthService run="true"/>
</requestMessage>
```

Requesting a Sale

You can request multiple services by adding additional elements. For example, if you fulfill the order immediately, you can request a credit card authorization and capture together (referred to as a "sale"):

```
<requestMessage xmlns="urn:schemas-cybersource-com:transaction-data-
1.18">
  <merchantID>infodev</merchantID>
  <ccAuthService run="true"/>
  <ccCaptureService run="true"/>
</requestMessage>
```

Adding Service-Specific Fields to the Request

You next add the fields that are used by the services you are requesting. Most fields are child elements of container elements; for example, a `<card>` element contains the customer's credit card information.

```
<requestMessage xmlns="urn:schemas-cybersource-com:transaction-data-
1.18">
  <merchantID>infodev</merchantID>
  <billTo>
    <firstName>Jane</firstName>
    <lastName>Smith</lastName>
  </billTo>
  <item id="0">
    <unitPrice>29.95</unitPrice>
  </item>
  <card>
    <accountNumber>4111111111111111</accountNumber>
  </card>
  <ccAuthService run="true"/>
</requestMessage>
```

The example above shows only a partial list of the fields you must send. Refer to "Related Documents," page 21 for information about the guides that list all of the fields for the services that you are requesting.

Sending Requests

Once you have created an XML document, you use Perl to send the request to CyberSource.

Adding the Use Statement

First add the use statement for the `cybs` module (for Linux) or the `CyberSource::SOAPI` module: (for Windows)

```
use CyberSource::SOAPI;
```

Loading the Configuration Settings

First load the configuration settings from a file:

```
%config = cybs_load_config( 'cybs.ini' );
```

Note The namespace that you specify in the XML document must use the same API version that you specify in the configuration settings file. For example, if `targetAPIVersion=1.18` in the file, the namespace must be `urn:schemas-cybersource-com:transaction-data-1.18`. The example code below retrieves the API version from the configuration settings file and places it in the XML document.

Reading the XML Document

```
# Read the XML document.
open IN, "auth.xml";
@req_array = <IN>;
close IN;
$inputXML = join( "", @req_array );

# Retrieve the target API version from the $config hash and replace the
# value in the XML document.
$inputXML
  =~ s/_APIVERSION_/$config{+CYBS_C_TARGET_API_VERSION}/;
```

Sending the Request

You next create the request hash, add the XML document to the hash, and send the request:

```
my %request;
# To use fully qualified constant names, see the important note in "Creating a Request
Document," page 332.
$request{+CYBS_SK_XML_DOCUMENT} = $inputXML;
# send request
my(%reply, $status);
$status = cybs_run_transaction( \%config, \%request, \%reply );
```

Interpreting Replies

Handling the Return Status

The `$status` value is the handle returned by the cybs_run_transaction() method. The `$status` indicates whether the CyberSource server received the request, the client received the reply, or there were any errors or faults during transmission. See "Possible Return Status Values," page 316 for descriptions of each status value. For a different example, see the `authSample.pl` file in the client's `<installation directory>/samples/xml` directory.

```
# To use fully qualified constant names, see the important note in "Creating a Request
Document," page 332.
if ($status == CYBS_S_OK) {
  # Read the value of the "decision" in the %reply hash.
  # This code assumes you have a method called getField() that retrieves the
  # specified field from the XML document in $reply{+CYBS_SK_XML_DOCUMENT}
  $decision = getField( \%reply, 'decision' );
  # If decision=ACCEPT, indicate to the customer that the request was successful.
  # If decision=REJECT, indicate to the customer that the order was not approved.
  # If decision=ERROR, indicate to the customer that an error occurred and to try
  # again later.
  # Now get reason code results:
  # $strContent = getReplyContent(\%reply);
  # See "Processing the Reason Codes," page 326 for how to process the reasonCode
  # from the reply.
  # Note that getReplyContent() is included in this document to help you understand
  # how to process reason codes, but it is not included as part of the sample scripts
  # or sample web pages.
}

else {

  handleError( $status, \%request, \%reply );
}

#---------------------
sub handleError
#---------------------
{
  # handleError shows how to handle the different errors that can occur.
  my ($nStatus, $pRequest, $pReply) = @_;

  # There was a problem with the parameters passed to cybs_run_transaction()
  if ( $nStatus == CYBS_S_PERL_PARAM_ERROR ) {
    # Non-critical error.
    # Tell customer the order cannot be completed and to try again later.
    # Notify appropriate internal resources of the error.
  }
```

```
# An error occurred before the request could be sent.
elsif ( $nStatus == CYBS_S_PRE_SEND_ERROR ) {
  # Non-critical error.
  # Tell customer the order cannot be completed and to try again later.
  # Notify appropriate internal resources of the error.
}

# An error occurred while sending the request.
elsif ( $nStatus == CYBS_S_SEND_ERROR ) {
  # Non-critical error.
  # Tell customer the order cannot be completed and to try again later.
}
# An error occurred while waiting for or retrieving the reply.
elsif ( $nStatus == CYBS_S_RECEIVE_ERROR ) {
  # Critial error.
  # Tell customer the order cannot be completed and to try again later.

  # Notify appropriate internal resources of the error.
  # See the sample code for more information about handling critical errors.
}
# An error occurred after receiving and during processing of the reply.
elsif ( $nStatus == CYBS_S_POST_RECEIVE_ERROR ) {
  # Critical error.
  # Tell customer the order cannot be completed and to try again later.
  # Look at CYBS_SK_RAW_REPLY in $reply for the raw reply.
  # Notify appropriate internal resources of the error.
  # See the sample code for more information about handling critical errors.
}

# CriticalServerError fault
elsif ( $nStatus == CYBS_S_CRITICAL_SERVER_FAULT ) {

  # Critial error.
  # Tell customer the order cannot be completed and to try again later.
  # Read the various fault details from the $reply.
  # Notify appropriate internal resources of the fault.
  # See the sample code for more information about reading fault details and
  # handling a critical error.
}
# ServerError fault
elsif ( $nStatus == CYBS_S_SERVER_FAULT ) {
  # Non-critical error.
  # Tell customer the order cannot be completed and to try again later.
  # Read the various fault details from the $reply.
  # See the sample code for information about reading fault details.
}
```

```
# Other fault
elsif ( $nStatus == CYBS_S_OTHER_FAULT ) {
  # Non-critical error.
  # Tell customer the order cannot be completed and to try again later.
  # Read the various fault details from the $reply.
  # Notify appropriate internal resources of the fault.
  # See the sample code for information about reading fault details.
}
```

Processing the Reason Codes

After the CyberSource server processes your request, it sends a reply message that contains information about the services you requested. You receive different fields depending on the services you request and the outcome of each service.

To use the reply information, you must integrate it into your system and any other system that uses that data. For example, you can store the reply information in a database and send it to other back office applications.

You must write an error handler to process the reply information that you receive from CyberSource. Do not show the reply information directly to customers. Instead, present an appropriate response that tells customers the result.

Important Because CyberSource may add reply fields and reason codes at any time, you should parse the reply data according to the names of the fields instead of their order in the reply.

The most important reply fields to evaluate are the following:

- **decision**: A one-word description of the results of your request. The decision is one of the following:
  - `ACCEPT` if the request succeeded
  - `REJECT` if one or more of the services in the request was declined
  - `REVIEW` if you use CyberSource Decision Manager and it flags the order for review. See "Handling Decision Manager Reviews," page 340 for more information.
  - `ERROR` if there was a system error. See "Retrying When System Errors Occur," page 330 for more information.
- **reasonCode**: A numeric code that provides more specific information about the results of your request.

You also receive a reason code for each service in your request. You can use these reason codes to determine whether a specific service succeeded or failed. If a service fails, other services in your request may not run. For example, if you request a credit card authorization and capture, and the authorization fails, the capture does not run. The reason codes for each service are described in the *Credit Card Services User Guide* for CyberSource Essentials merchants or in the service developer guide for CyberSource Advanced merchants.

Important CyberSource reserves the right to add new reason codes at any time. If your error handler receives a reason code that it does not recognize, it should use the decision to interpret the reply.

```
# Note that getReplyContent is included in this document to help you understand how
# to process reason codes, but it is not included as part of the sample scripts or
# sample web pages.

# This code assumes you have a method called getField() that retrieves the specified
# field from the XML document in $reply{+CYBS_SK_XML_DOCUMENT}.
#----------------
sub getReplyContent( $pReply )
#----------------
{
  my ($pReply) = @_;
  $reasonCode = $$pReply{'reasonCode'};

  # Success
  if ($reasonCode == '100'){
    return( sprintf(
      "Request ID: %s\nAuthorizedAmount: %s\nAuthorization Code: %s,
      $$pReply{'requestID'}, $$pReply{'ccAuthReply_amount'},
      $$pReply{'ccAuthReply_authorizationCode'} ) );
  }

  # Insufficient funds
  elsif ($reasonCode == '204'){
    return( sprintf(
    "Insufficient funds in account. Please use a different
      card or select another form of payment." ) );
  }
    # add other reason codes here that you must handle specifically
  else {
    # For all other reason codes, return NULL, in which case, you should display a
    # generic message appropriate to the decision value you received.
    return( NULL );
  }
}
```

Handling Decision Manager Reviews

If you use CyberSource Decision Manager, you may also receive the `REVIEW` value in the **decision** field. `REVIEW` means that Decision Manager has marked the order for review based on how you configured the Decision Manager rules.

If you will be using Decision Manager, you have to determine how to handle the new `REVIEW` value. Ideally, you will update your order management system to recognize the `REVIEW` response and handle it according to your business rules. If you cannot update your system to handle the `REVIEW` response, CyberSource recommends that you choose one of these options:

- If you authorize and capture the credit card payment at the same time, treat the `REVIEW` response like a `REJECT` response. Rejecting any orders that are marked for review may be appropriate if your product is a software download or access to a Web site. If supported by your processor, you may also want to reverse the authorization.

- If you approve the order after reviewing it, convert the order status to `ACCEPT` in your order management system. You can request the credit card capture without requesting a new authorization.

- If you approve the order after reviewing it but cannot convert the order status to `ACCEPT` in your system, request a new authorization for the order. When processing this new authorization, you must disable Decision Manager. Otherwise the order will be marked for review again. For details about the API field that disables Decision Manager, see the *Decision Manager Developer Guide Using the Simple Order API* (PDF | HTML) or the *Decision Manager Developer Guide Using the SCMP Order API* (PDF | HTML).

Alternately, you can specify a custom business rule in Decision Manager so that authorizations originating from a particular internal IP address at your company are automatically accepted.

If supported by your processor, you may want to reverse the original authorization.

Requesting Multiple Services

When you request multiple services in one request, CyberSource processes the services in a specific order. If a service fails, CyberSource does not process the subsequent services in the request.

For example, in the case of a sale (a credit card authorization and a capture requested together), if the authorization service fails, CyberSource will not process the capture service. The reply you receive only includes reply fields for the authorization.

This following additional example applies **to CyberSource Advanced merchants only.**

Many CyberSource services include “ignore” fields that tell CyberSource to ignore the result from the first service when deciding whether to run the subsequent services. In the case of the sale, even though the issuing bank gives you an authorization code, CyberSource might decline the authorization based on the AVS or card verification results. Depending on your business needs, you might choose to capture these types of declined authorizations anyway. You can set the **businessRules_ignoreAVSResult** field to “true” in your combined authorization and capture request:

```
<businessRules>
    <ignoreAVSResult>true</ignoreAVSResult>
</businessRules>
```

This tells CyberSource to continue processing the capture even if the AVS result causes CyberSource to decline the authorization. In this case you would then get reply fields for both the authorization and the capture in your reply.

Note You are charged only for the services that CyberSource performs.

Retrying When System Errors Occur

You must design your transaction management system to include a way to correctly handle CyberSource system errors. Depending on which payment processor is handling the transaction, the error may indicate a valid CyberSource system error, or it may indicate a processor rejection because of some type of invalid data. In either case, CyberSource recommends that you do not design your system to retry sending a transaction many times in the case of a system error.

Instead, CyberSource recommends that you retry sending the request only two or three times with successively longer periods of time between each retry. For example, after the first system error response, wait 30 seconds and then retry sending the request. If you receive the same error a second time, wait one minute before you send the request again. Depending on the situation, you may decide you can retry sending the request after a longer time period. Determine what is most appropriate for your business situation.

If after several retry attempts you are still receiving a system error, it is possible that the error is actually being caused by a processor rejection and not a CyberSource system error. In that case, we suggest that you either:

- Search for the transaction in the Business Center, look at the description of the error on the Transaction Detail page, and call your processor to determine if and why they are rejecting the transaction.
- Contact CyberSource Customer Support to confirm whether your error is truly caused by a CyberSource system issue.

If TSYS Acquiring Solutions is your processor, you may want to follow the first suggestion as there are several common TSYS Acquiring Solutions processor responses that are returned to you as system errors and that only TSYS Acquiring Solutions can address.

Advanced Configuration Information

Using Alternate Server Configuration Settings

You use the serverURL and namespaceURI configuration settings if CyberSource changes the convention we use to specify the server URL and namespace URI, but we have not had the opportunity to update the client yet.

For example, these are the server URLs and namespace URI for accessing the CyberSource services using the Simple Order API version 1.18:

- Test server URL:

  `https://ics2wstesta.ic3.com/commerce/1.x/transactionProcessor`

- Production server URL:

  `https://ics2wsa.ic3.com/commerce/1.x/transactionProcessor`

- Namespace URI:

  `urn:schemas-cybersource-com:transaction-data-1.18.`

Note If you view the above URLs with a web browser, a list of the supported API versions and the associated schema files are displayed.

If in the future CyberSource changes these conventions, but does not provide a new version of the client, you can configure your existing client to use the new server and namespace conventions required by the CyberSource server.

Configuring for Multiple Merchant IDs

If you have multiple merchant IDs, or if you are a reseller handling multiple merchants, you can have different configuration settings for different merchant IDs. You set these in the configuration object that you pass to the cybs_run_transaction() function. When using the samples provided in the client package, you set the configuration parameters in `cybs.ini` file.

All of the properties except merchantID can be prefixed with "`<merchantID>.`" to specify the settings for a specific merchant.

Example Merchant-Specific Properties Settings

If you have a merchant with merchant ID of `merchant123`, and you want enable logging only for that merchant, you can set the `enableLog` parameter to `true` for all requests that have `merchant123` as the merchant ID:

```
merchant123.enableLog=true
enableLog=false
```

The client disables logging for all other merchants.

CHAPTER 9

PHP Client

Important

- This chapter covers the Linux and Windows platforms and uses the Linux convention of forward slashes when path names are listed.
- The PHP client for the Simple Order API is supported on 32-bit operating systems only.
- If you are building an application to sell to others, see Appendix A, "Using the Client Application Fields," on page 390. This appendix has a list of API fields you can use in your request that describe the application, its version, and its user. If you use these fields in your request, you can view their values in the Transaction Search Details window of the Business Center.

Using PHP in a Hosted Environment

If you are operating in a hosted environment (with an Internet Service Provider hosting your web store), read this section.

To use the CyberSource Simple Order API client for PHP, you must register a PHP extension in `php.ini` and modify the LD_LIBRARY_PATH (for Linux) or the system PATH (for Windows) to include the `lib` directory of the CyberSource client. The CyberSource binaries ensure that your transactions are secure while being sent to CyberSource. If you use a hosted environment, you must check with your hosting provider (ISP) to make sure that they support the addition of a PHP extension and editing of the path environment variables.

If you cannot find any documentation related to your hosting provider's support of extensions and new library locations, contact your hosting provider with this statement:

> **CyberSource requires modifying php.ini to add their extension and editing of LD_LIBRARY_PATH (for Linux) or the system PATH (for Windows) to add the directory containing the dynamic libraries required by the extension for use by my e-commerce software. CyberSource ensures the safety and functionality of these libraries. Please let me know your policy for supporting this implementation.**

Because other merchants who use your hosting provider may also use CyberSource, your hosting provider may have already installed the CyberSource PHP client. In that case, we suggest that you verify with your hosting provider the version of the client they have installed and registered. If the client you want to use is newer, ask them to replace the libraries with the new ones.

If you have any questions regarding the above information or installation of the client, please contact Customer Support. If you are a Business Center user, and you cannot obtain the appropriate access from your ISP to install the client, consider using CyberSource's Hosted Order Page or Simple Order Post instead of the PHP client. These connection methods are described in the *Hosted Order Page User's Guide* and the *Silent Order Post User's Guide*, both of which are available in the Business Center.

Choosing Your API and Client

API Variation

With this client package, you can use either of these variations of the Simple Order API:

- Name-value pairs, which are simpler to use than XML
- XML, which requires you to create and parse XML documents

The test that you run immediately after installing the client uses name-value pairs.

Client Versions

CyberSource regularly updates the Simple Order API to introduce new API fields and functionality. To identify the latest version of the server-side API for the CyberSource services, go to https://ics2wsa.ic3.com/commerce/1.x/transactionProcessor.

The Simple Order API Client for PHP also has a version, but it is not the same as the API version. The client version represents the version of the client-side code that you use to access the CyberSource services.

When configuring the client, you indicate the version of the API that you want to use. When setting this parameter, do not use the current version of the client; use the current version of the API.

Sample Code

The client contains sample scripts and sample PHP pages that you can use to test the client.

Basic PHP Page Example

The example below shows the primary code required to send a Simple Order API request for credit card authorization. The example uses name-value pairs. For a more complete example, see the sample program and sample PHP pages included in the package (see "Sample Code," page 347). "Using Name-Value Pairs," page 366 shows you how to create the code.

```
// Load the configuration settings
$config = cybs_load_config( 'cybs.ini' );

// set up the request by creating an array and adding fields to it
$request = array();

// We want to do credit card authorization in this example
$request['ccAuthService_run'] = "true";
// Add required fields
$request['merchantID'] = 'infodev';
$request['merchantReferenceCode'] = 'MRC-14344';
$request['billTo_firstName'] = 'Jane';
$request['billTo_lastName'] = 'Smith';
$request['billTo_street1'] = '1295 Charleston Road';
$request['billTo_city'] = 'Mountain View';
$request['billTo_state'] = 'CA';
$request['billTo_postalCode'] = '94043';
$request['billTo_country'] = 'US';
$request['billTo_email'] = 'jsmith@example.com';
$request['card_accountNumber'] = '4111111111111111';
$request['card_expirationMonth'] = '12';
$request['card_expirationYear'] = '2010';
$request['purchaseTotals_currency'] = 'USD';

// This example has two items
$request['item_0_unitPrice'] = '12.34';
$request['item_1_unitPrice'] = '56.78';

// Add optional fields here according to your business needs

// Send request
$reply = array();
$status = cybs_run_transaction( $config, $request, $reply );
// Handle the reply. See "Handling the Return Status," page 369.
```

Sample Scripts

The client contains two sample scripts, one for using name-value pairs and one for using XML. See "Testing the Client," page 356 or see the README file for more information about using the `authCaptureSample.php` script to test the client.

- Name-value pairs: See `authCaptureSample.php` in *<installation directory>*`/samples/nvp`.
- XML: We suggest that you examine the name-value pair sample code listed above before implementing your code to process XML requests.

For the XML sample code, see `authSample.php` in *<installation directory>*`/samples/xml`. Also see the `auth.xml` XML document that the script uses.

Sample PHP Pages

The client download package also includes sample PHP pages in the *<installation directory>*`/samples/store` directory.

Table 43 Files in sampleStore Directory

File	Description
`util.php`	Used by the other PHP pages in the directory.
`checkout.php`	Displays the contents of the shopping basket and prompts for address and payment information.
`checkout2.php`	Authorizes the order and displays the result.
`store_footer.php`	Footer used in the checkout pages.
`store_header.php`	Header used in the checkout pages.

To use the sample PHP pages:

Step 1 If you have files in your web server's root directory that have the same name as the files listed in Table 43, "Files in sampleStore Directory," on page 348, back up those files.

You will be copying the sample store files into the root directory in the next step. For Apache, the root directory is the one specified by `DocumentRoot` in `httpd.conf`.

Step 2 Copy all of the files in the *<installation directory>*`/samples/store` directory into your web server's root directory.

Step 3 Modify the `cybs.ini` file as appropriate. For more information, see "Configuring Client Settings," page 354.

Important

Use absolute paths for the directories in the `cybs.ini` file that you use with the sample store, for example: `keysDirectory=c:\keys`.

If you encounter problems getting the sample PHP pages to work, you might need to locate your `cybs.ini` file outside of the root directory.

Step 4 Open the `checkout.php` file in a text editor and locate the `cybs_load_config()` function.

Step 5 Make sure that the parameter for the `cybs.ini` file passed to the function includes the absolute path. For example, make sure the line reads:

```
$config = cybs_load_config( 'c:\cybs.ini' );
```

not this line:

```
$config = cybs_load_config( 'cybs.ini' );
```

Step 6 Restart your web server.

If you are using Microsoft Internet Information Services (IIS), you might need to restart your computer for IIS to pick up the new server path.

Step 7 Open a web browser and type the following URL:

`http://`*`<your web server name or IP address>`*`/`*`<virtual directory if applicable>`*`/checkout.php`

Installing and Testing the Client

Minimum System Requirements

For Linux

- Linux kernel 2.2, LibC6 on an Intel processor (for RedHat users, this currently corresponds to versions 7.1 and 7.2)
- PHP4 (minimum version 4.2.1) or PHP5 (5.0.0–5.0.3 and 5.1.0-5.1.2)
- GNU GCC

For Windows

- Windows XP, 2000, or newer
- Minimum PHP version 4.2.1

The SDK supports UTF-8 encoding.

Important Failure to configure your client API host to a unique, public IP address will cause inconsistent transaction results.

The client API request ID algorithm uses a combination of IP address and system time, along with other values. In some architectures this combination might not yield unique identifiers.

Transaction Security Keys

The first thing you must do is create your security key. The client uses the security key to add a digital signature to every request that you send. This signature helps ensure that no one else can use your CyberSource account to process orders. You specify the location of your key when you configure the client.

Important You must generate two transaction security keys—one for the CyberSource production environment and one for the test environment. For information about generating and using security keys, see *Creating and Using Security Keys* (PDF | HTML).

The Simple Order API client for PHP package includes the `ca-bundle.crt`, a bundle of certificate files. The client expects to find the `ca-bundle.crt` file in the same directory as your security keys. If you move it elsewhere, use the sslCertFile configuration parameter to specify the file location. For more information, see the description of the parameter "sslCertFile," page 355.

Warning You must protect your security key to ensure that your CyberSource account is not compromised.

Installing the Client

This section describes the installation steps for Linux and Windows environments.

To install the client on Linux:

Step 1 Go to the client downloads page on the Support Center.

Step 2 Download the latest client package. You can save the file in any directory.

Step 3 Unzip and untar the package.

This creates a directory called `simapi-php-n.n.n`, where `n.n.n` is the client version.

Important The `simapi-php-n.n.n/lib` directory contains symbolic links. If you install the client by copying the `lib` directory from some other location where you untarred the package, check to see if the symbolic links are still there. If they are not, you must recreate them.

Step 4 Copy the `phpN_cybersource.so` file into the PHP extension directory, where the ***N*** is 4 if your PHP version is 4.x.x; 5 if your PHP version is 5.0.0-5.0.2; 503 if your PHP version is 5.0.3.; or 512 if your version is 5.1.0-5.1.2.

The extension directory is the one `"extension_dir"` is set to in the `php.ini` file. If you do not already have `"extension_dir"` set to an explicit directory:

a Create an extension directory (outside of the client installation directory).

b Set `"extension_dir"` to that directory.

c Copy the `php`***N***`_cybersource.so` file to that directory location.

Step 5 If you are using an Oracle database, go to "Special Installation Instructions for Oracle Users," page 359 and follow the instructions.

Otherwise, in the `php.ini` file, locate the “Dynamic Extensions” section and add one of the following lines anywhere before the next section in the file:

`extension=php4_cybersource.so` (if using PHP 4.x.x) or

`extension=php5_cybersource.so` (if using PHP 5.0.0-5.0.2)

`extension=php503_cybersource.so` (if using PHP 5.0.3) or

`extension=php512_cybersource.so` (if using PHP 5.1.0-5.1.2)

Step 6 Save the `php.ini` file.

Step 7 Modify the environment variable LD_LIBRARY_PATH to include the `lib` directory of the CyberSource client. For example:

```
export LD_LIBRARY_PATH=/baseDir/simapi-php-n.n.n/lib:$LD_LIBRARY_
PATH
```

where `/baseDir` is the directory where you untarred the CyberSource client package.

Note If the web server is running as the user "nobody", you must use `ldconfig` instead of setting the LD_LIBRARY_PATH. In this case, update the `/etc/ld.so.conf` file to include the library path (`/baseDir/simapi-php-n.n.n/lib`), and run `ldconfig` to update the configuration.

Step 8 Configure the client. See "Configuring Client Settings," page 354 below.

Step 9 Test the client. See "Testing the Client," page 356.

To install the client on Windows:

Step 1 Go to the client downloads page on the Support Center.

Step 2 Download the latest client package. You can save the file in any directory.

Step 3 Unzip the package.

This creates a directory called `simapi-php-n.n.n`, where `n.n.n` is the client version.

Step 4 Copy the `php`***`N`***`_cybersource.dll` file into the PHP extension directory, where the ***N*** is 4 if your PHP version is 4.x.x, or 5 if your PHP version is 5.x.x.

The extension directory is the one `"extension_dir"` is set to in the `php.ini` file. If you do not already have `"extension_dir"` set to an explicit directory:

a Create an extension directory (outside of the client installation directory).

b Set `"extension_dir"` to that directory.

c Copy the `php`***`N`***`_cybersource.dll` file to that directory location.

Step 5 In the `php.ini` file, locate the "Windows Extensions" section and add one of the following lines anywhere before the next section in the file:

`extension=php4_cybersource.dll` (if using PHP 4.x.x) or

`extension=php5_cybersource.dll` (if using PHP 5.0.0–5.0.2)

`extension=php503_cybersource.dll` (if using PHP 5.0.3) or

`extension=php512_cybersource.dll` (if using PHP 5.1.0-5.1.2)

Step 6 Save the `php.ini` file.

Step 7 Add the `lib` directory of the CyberSource client package to the system PATH. This makes the DLLs included in the client package available to the CyberSource PHP extension.

The client is installed on your system.

Step 8 Configure the client. See "Configuring Client Settings," page 354 below.

Step 9 Test the client. See "Testing the Client," page 356.

Configuring Client Settings

To run the sample scripts included in the client package, you must set the configuration parameters in the `cybs.ini` file, which is located in the `<installation directory>/samples` directory for Linux, and in the `nvp`, `xml`, and `store` subfolders inside the `samples` directory for Windows. You can also use this file when running transactions in a production environment (see the function descriptions in "PHP API for the Client," page 360). The following table describes the parameters that you can set. The default `cybs.ini` file that comes with the client package does not include all of the parameters listed in the table. The file includes only the parameters required to run the sample scripts.

If you have multiple merchant IDs, or if you are a reseller handling multiple merchants, you can use different configuration settings depending on the merchant ID. See "Configuring Your Settings for Multiple Merchant IDs," page 389 for more information.

Table 44 Configuration Settings

Setting	Description
merchantID	Merchant ID. The client uses this value if you do not specify a merchant ID in the request itself.
keysDirectory	Location of the merchant's security key. The client includes a `keys` directory that you can use. Include the path, for example: `../keys`, or `c:\simapi-php-1.0.0\keys`. **Note** We recommend that you store your key locally for faster request processing.
sendToProduction	Flag that indicates whether the transactions for this merchant should be sent to the production server. Use one of these values: ■ `false`: Do not send to the production server; send to the test server (default setting). ■ `true`: Send to the production server. **Note** Make sure that if your merchant ID is configured to use the test mode, you send requests to the test server.
targetAPIVersion	Version of the Simple Order API to use. **Note** For a current list of the available versions, go to https://ics2wsa.ic3.com/commerce/1.x/transactionProcessor. For information about what has changed in each version, see the *Simple Order API Release Notes.*
keyFilename	Name of the security key file name for the merchant in the format `<`***`security_key_filename`***`>.p12`.
serverURL	Alternate server URL to use. See "Using Alternate Server Configuration Settings," page 388 for more information. Give the complete URL because it will be used exactly as you specify here.
namespaceURI	Alternate namespace URI to use. See "Using Alternate Server Configuration Settings," page 388 for more information. Give the complete namespace URI because it will be used exactly as you specify here.

Table 44 Configuration Settings (Continued)

Setting	Description
enableLog	Flag directing the client to log transactions and errors. Use one of these values: ■ `false`: Do not enable logging (default setting). ■ `true`: Enable logging. **Important** Logging can cause very large log files to accumulate. Therefore, CyberSource recommends that you use logging only when troubleshooting problems. To comply with all Payment Card Industry (PCI) and Payment Application (PA) Data Security Standards regarding the storage of credit card and card verification number data, the logs that are generated contain only masked credit card and card verification number data (CVV, CVC2, CVV2, CID, CVN). Follow these guidelines: ■ Use debugging temporarily for diagnostic purposes only. ■ If possible, use debugging only with test credit card numbers. ■ Never store clear text card verification numbers. ■ Delete the log files as soon as you no longer need them. ■ Never send email to CyberSource containing personal and account information, such as customers' names, addresses, card or check account numbers, and card verification numbers. For more information about PCI and PABP requirements, see www.visa.com/cisp.
logDirectory	Directory to which to write the log file. Note that the client will not create this directory for you; you must specify an existing directory.The client includes a `logs` directory that you can use. Include the path, for example: `../logs`, or `c:\simapi-php-1.0.0\logs`.
logFilename	Log file name. The client uses `cybs.log` by default.
logMaximumSize	Maximum size in megabytes for the log file. The default value is `10`. When the log file reaches the specified size, it is archived into `cybs.log.`***<yyyymmdd***`T`***hhmmssxxx>*** and a new log file is started. The ***xxx*** indicates milliseconds.
sslCertFile	The location of the bundled file of CA Root Certificates (`ca-bundle.crt`) which is included in the client download package. The client automatically looks for the file in the directory where your security key is stored (specified by keysDirectory). If you move the file so it does not reside in keysDirectory, use this configuration setting to specify the full path to the file, including the file name.
timeout	Length of time-out in seconds. The default is 110.

Table 44 Configuration Settings (Continued)

Setting	Description
proxyServer	Proxy server to use. Allowable formats include: - *<http://>server<:port>* - *<http://>IP address<:port>* The `http://` and `port` are optional. **Note** The default port is 1080. If your proxy server is listening on another port, you must specify a port number.
proxyUsername	Username used to authenticate against the proxy server if required. If the proxy server requires the domain name during authentication, add the domain name and a backslash: ***<domain>***`\`***<username>***
proxyPassword	Password used to authenticate against the proxy server, if required.

Testing the Client

After you install and configure the client, test it immediately to ensure that the installation was successful.

To test the client:

Step 1 Go to the *<installation directory>*`/samples/nvp` directory.

Step 2 Run the test `authCaptureSample.php` script by typing:

```
php authCaptureSample.php
```

where **php** is the command-line interface (CLI) version. Depending on the PHP version, **php** may be in the main PHP directory, the `sapi/cli` directory, the `cli` directory, or it may be named `php-cli.exe` or `php.exe`.

For example, for PHP 4.3.0 with Linux, you might have:

```
<PHP directory>/sapi/cli/php authCaptureSample.php
```

Or for PHP 4.3.8 with Windows, you might have:

```
<PHP directory>\cli\php authCaptureSample.php
```

or

```
<PHP directory>\php.exe authCaptureSample.php
```

The results of the test are displayed in the window.

- If the test is successful, a decision of ACCEPT appears for both the credit card authorization and the follow-on capture.
- If the test is not successful, a different decision value or an error message appears.

To troubleshoot client test failures:

Step 1 Verify that your `cybs.ini` settings are correct.

Step 2 Run the test again.

Step 3 If the test still fails, look at the error message and determine the return status value (a numeric value from -1 to 8).

Step 4 See the descriptions of the status values in "Possible Return Status Values," page 362, and follow any instructions given there for the error you received.

Step 5 Run the test again.

Step 6 If the test still fails, contact Customer Support.

To run the XML sample:

Step 1 Go to the `<`*`installation directory`*`>/sample/xml` directory.

Step 2 For Windows, modify the `cybs.ini` in the folder with your settings (for Linux, make sure the `samples/cybs.ini` file is set how you want it).

Step 3 Run the test `authSample.php` script by typing:

```
php authSample.php
```

The results of the test are displayed in the window.

- If the test is successful, you see a decision of ACCEPT for both the credit card authorization and the follow-on capture.
- If the test is not successful, you see a different decision value or an error message. See "To troubleshoot client test failures:," page 357 to troubleshoot the error.

The client is installed and tested. You are ready to create your own code for requesting CyberSource services. For information about creating requests, see "Using Name-Value Pairs," page 366 if you plan to use name-value pairs, or "Using XML," page 376 if you plan to use XML.

Going Live

When you complete all of your system testing and are ready to accept real transactions from your customers, your deployment is ready to *go live*.

Note

After your deployment goes live, use real card numbers and other data to test every card type you support. Because these are real transactions in which you are buying from yourself, use small monetary amounts to do the tests. Process an authorization, then capture the authorization, and later refund the money. Use your bank statements to verify that money is deposited into and withdrawn from your merchant bank account as expected. If you have more than one CyberSource merchant ID, test each one separately.

CyberSource Essentials Merchants

If you use CyberSource Essentials services, you can use the Business Center site to go live. For a description of the process of going live, see the “Steps for Getting Started” section in *Getting Started with CyberSource Essentials*.

Important

Configure your client so that it can send transactions to the production server and not the test server. For more information, see the description of the configuration setting "sendToProduction," page 354.

CyberSource Advanced Merchants

If you use CyberSource Advanced services, see the “Steps for Getting Started” chapter in *Getting Started with CyberSource Advanced* for information about going live.

When your deployment goes live, your CyberSource account is updated so that you can send transactions to the CyberSource production server. If you have not already done so, you must provide your banking information to CyberSource so that your processor can deposit funds to your merchant bank account.

After CyberSource has confirmed that your account is live, make sure that you update your system so that it can send requests to the production server (`ics2wsa.ic3.com`) using your security key for the production environment. The test server (`ics2wstesta.ic3.com`) cannot be used for real transactions. For more information about sending transactions to the production server, see the description of the configuration setting "sendToProduction," page 354.

Updating the Client to Use a Later API Version

CyberSource periodically updates the Simple Order API (previously called the Web Services API). You can update your existing client to work with the new API version. Go to https://ics2wsa.ic3.com/commerce/1.x/transactionProcessor for a list of the available API versions. Or, if you are in test mode, go to https://ics2wstesta.ic3.com/commerce/1.x/transactionProcessor.

To update the client to use a later API version, update the value for the `targetAPIVersion` configuration parameter in the `cybs.ini` file. For example, to use the 1.18 version of the API, set the property to `1.18`.

Special Installation Instructions for Oracle Users

If you are using Linux and an Oracle database, you must:

- Load the Oracle extensions dynamically
- In the `php.ini` file, load the CyberSource extension before the Oracle extensions

To load Oracle extensions dynamically after the CyberSource extension:

Step 1 At a command prompt, go to your PHP directory.

Step 2 Type the following:

```
make clean
```

Step 3 Execute **`configure`** so that you are loading the Oracle extensions dynamically. To do this, include “**`shared,`**” before the path to each Oracle extension. For example, you might execute **`configure`** as follows:

```
./configure --prefix=<target PHP directory>
--with-apxs=/usr/local/apache_1.3.32/bin/apxs
--with-oracle=shared,/home/u01/app/oracle/product/8.1.7
--with-oci8=shared,/home/u01/app/oracle/product/8.1.7
--without-mysql
```

Step 4 Type the following:

```
make
make install
```

Step 5 In the “Dynamic Extensions” section of the `php.ini` file, add the CyberSource extension before the Oracle extensions:

`extension=phpN_cybersource.so` (where ***N*** represents the version of PHP: 4, 5, 503, or 512)

```
extension = oracle.so
```

```
extension = oci8.so
```

Step 6 Save the `php.ini` file.

Step 7 Continue with the original installation instructions (see Step 7 on page 352).

PHP API for the Client

Summary of Functions

The client includes these functions:

- cybs_load_config()
- cybs_run_transaction()

cybs_load_config()

Table 45 cybs_load_config()

Syntax	`array cybs_load_config( string filename )`
Description	Loads the configuration settings from a file
Returns	An array containing the configuration settings
Parameters	`filename`: Name of the configuration file

cybs_run_transaction()

Table 46 cybs_run_transaction()

Syntax	`int cybs_run_transaction( array config, array request, array reply )`	
Description	Sends the request to the CyberSource server and receives the reply	
Returns	A value that indicates the status of the request	
Parameters	`config`: Configuration array to use	
	`request`: Array containing one of these:	■ The individual name-value pairs in the request (for name-value pair users) ■ A single key called `CYBS_SK_XML_DOCUMENT` whose value is the XML document representing the request (for XML users)

	`reply`:
	Array containing one of these:
		Note You must create this array before you call cybs_run_transaction().
		■ The individual name-value pairs in the reply (for name-value pair users)
		■ A single key called `CYBS_SK_XML_DOCUMENT` whose value is the XML document representing the reply (for XML users)
		■ A combination of the following keys and their values:
		`CYBS_SK_ERROR_INFO`
		`CYBS_SK_RAW_REPLY`
		`CYBS_SK_FAULT_DOCUMENT`
		`CYBS_SK_FAULT_CODE`
		`CYBS_SK_FAULT_STRING`
		`CYBS_SK_FAULT_REQUEST_ID`
		See below for descriptions of these keys.

Reply Key Descriptions

- `CYBS_SK_ERROR_INFO`: Information about the error that occurred
- `CYBS_SK_RAW_REPLY`: The server's raw reply
- `CYBS_SK_FAULT_DOCUMENT`: The entire, unparsed fault document
- `CYBS_SK_FAULT_CODE`: The fault code, which indicates where the fault originated
- `CYBS_SK_FAULT_STRING`: The fault string, which describes the fault
- `CYBS_SK_FAULT_REQUEST_ID`: The request ID for the request

Possible Return Status Values

The cybs_run_transaction() function returns a status indicating the result of the request. Table 47, "Possible Status Values," on page 362 describes the possible status values, including whether the error is critical. If an error occurs after the request has been sent to the server, but the client cannot determine whether the transaction was successful, then the error is considered critical. If a critical error happens, the transaction may be complete in the CyberSource system but not complete in your order system. The descriptions below indicate how to handle critical errors.

Note

The sample scripts display a numeric value for the return status, which is listed in the first column.

Table 47 Possible Status Values

Numeric Value (for Sample Scripts)	Value	Description
0	CYBS_S_OK	**Critical:** No **Result:** The client successfully received a reply. For name-value pair users, the `$reply` array has the reply name-value pairs for the services that you requested. For XML users, the `$reply` array contains the response in XML format. **Manual action to take:** None
-1	CYBS_S_PHP_ PARAM_ERROR	**Critical:** No **Result:** The request was not sent because there was a problem with one or more of the parameters passed to the cybs_run_transaction() function. **Manual action to take:** Make sure the parameter values are correct.
1	CYBS_S_PRE_SEND_ ERROR	**Critical:** No **Result:** An error occurred before the request could be sent. This usually indicates a configuration problem with the client. **Error information to read:** `$reply[CYBS_SK_ERROR_INFO]` **Manual action to take:** Fix the problem described in the error information.

Table 47 Possible Status Values (Continued)

Numeric Value (for Sample Scripts)	Value	Description
2	CYBS_S_SEND_ ERROR	**Critical:** No **Result:** An error occurred while sending the request. **Error information to read:** `$reply[CYBS_SK_ERROR_INFO]` **Manual action to take:** None **Note** A typical send error that you might receive when testing occurs if the `ca-bundle.crt` file is not located in the same directory as your security key. For information about how to fix the problem, see the description of the configuration parameter "sslCertFile," page 355.

3	CYBS_S_RECEIVE_ ERROR	**Critical:** Yes
		Result: An error occurred while waiting for or retrieving the reply.
		Error information to read:
		`$reply[CYBS_SK_ERROR_INFO]`
		`$reply[CYBS_SK_RAW_REPLY]`
		Manual action to take: Check the Transaction Search screens on the Business Center to verify that the request was processed, and if so, whether it succeeded. Update your transaction database appropriately.

4	CYBS_S_POST_ RECEIVE_ERROR	**Critical:** Yes
		Result: The client received a reply or a fault, but an error occurred while processing it.
		Error information to read:
		`$reply[CYBS_SK_ERROR_INFO]`
		`$reply[CYBS_SK_RAW_REPLY]`
		Manual action to take: Examine the value of `$reply[CYBS_SK_RAW_REPLY]`. If you cannot determine the status of the request, then check the Transaction Search screens on the Business Center to verify that the request was processed, and if so, whether it succeeded. Update your transaction database appropriately.

Table 47 Possible Status Values (Continued)

Numeric Value (for Sample Scripts)	Value	Description

5	CYBS_S_CRITICAL_SERVER_FAULT	**Critical:** Yes
		Result: The server returned a fault with `$reply[CYBS_SK_FAULT_CODE]` set to CriticalServerError.
		Error information to read:
		`$reply[CYBS_SK_ERROR_INFO]`
		`$reply[CYBS_SK_FAULT_CODE]`
		`$reply[CYBS_SK_FAULT_STRING]`
		`$reply[CYBS_SK_FAULT_DOCUMENT]`
		`$reply[CYBS_SK_FAULT_REQUEST_ID]`
		Manual action to take: Check the Transaction Search screens on the Business Center to verify that the request succeeded. When searching for the request, use the request ID provided by `$reply[CYBS_SK_FAULT_REQUEST_ID]`.

6	CYBS_S_SERVER_FAULT	**Critical:** No
		Result: The server returned a fault with `$reply[CYBS_SK_FAULT_CODE]` set to ServerError, indicating a problem with the CyberSource server.
		Error information to read:
		`$reply[CYBS_SK_ERROR_INFO]`
		`$reply[CYBS_SK_FAULT_CODE]`
		`$reply[CYBS_SK_FAULT_STRING]`
		`$reply[CYBS_SK_FAULT_DOCUMENT]`
		Manual action to take: None

Table 47 Possible Status Values (Continued)

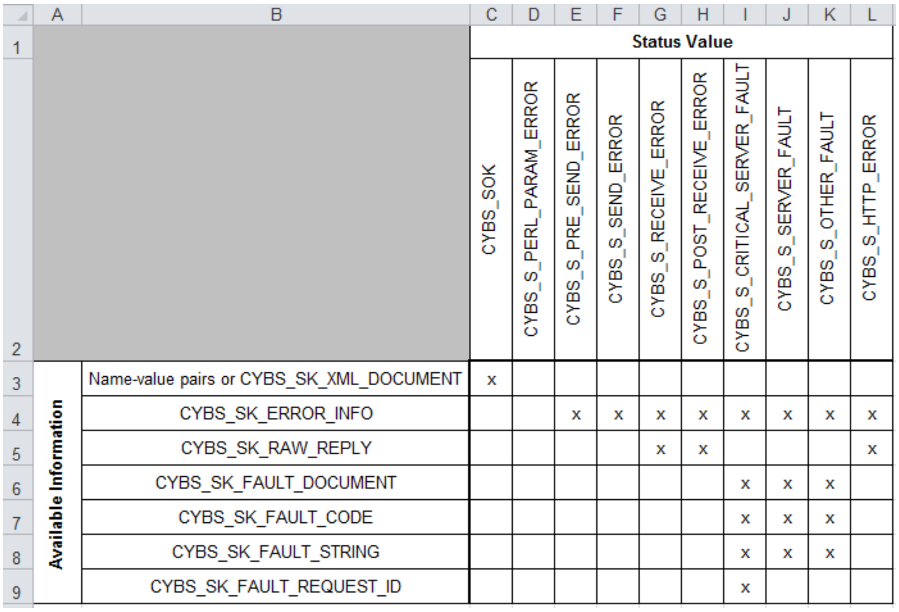
Numeric Value (for Sample Scripts)	Value	Description

7	CYBS_S_OTHER_FAULT	**Critical:** No
		Result: The server returned a fault with `$reply[CYBS_SK_FAULT_CODE]` set to a value other than ServerError or CriticalServerError. Indicates a possible problem with merchant status or the security key. Could also indicate that the message was tampered with after it was signed and before it reached the CyberSource server.
		Error information to read:
		`$reply[CYBS_SK_ERROR_INFO]`
		`$reply[CYBS_SK_FAULT_CODE]`
		`$reply[CYBS_SK_FAULT_STRING]`
		`$reply[CYBS_SK_FAULT_DOCUMENT]`
		Manual action to take: Examine the value of the `$reply[CYBS_SK_FAULT_STRING]` and fix the problem. You might need to generate a new security key, or you might need to contact Customer Support if there are problems with your merchant status.
		Note A typical error that you might receive occurs if your merchant ID is configured for “test” mode but you send transactions to the production server. For information about fixing the problem, see the description of the configuration parameter "sendToProduction," page 354.

8	CYBS_S_HTTP_ERROR	**Critical:** No
		Result: The server returned an HTTP status code other than 200 (OK) or 504 (gateway timeout). Note that if a 504 gateway timeout occurs, then the status=`3`.
		Error information to read:
		`$reply[CYBS_SK_ERROR_INFO]`
		`$reply[CYBS_SK_RAW_REPLY]`
		Value of varReply: `CYBS_SK_RAW_REPLY` contains the HTTP response body, or if none was returned, the literal `"(no response available)"`.
		Manual action to take: None.

Table 48 summarizes which reply information you receive for each status value.

Table 48 Reply Information Available for Each Status Value



	A	B	C	D	E	F	G	H	I	J	K	L
1			**Status Value**									
2			CYBS_SOK	CYBS_S_PERL_PARAM_ERROR	CYBS_S_PRE_SEND_ERROR	CYBS_S_SEND_ERROR	CYBS_S_RECEIVE_ERROR	CYBS_S_POST_RECEIVE_ERROR	CYBS_S_CRITICAL_SERVER_FAULT	CYBS_S_SERVER_FAULT	CYBS_S_OTHER_FAULT	CYBS_S_HTTP_ERROR
3	**Available Information**	Name-value pairs or CYBS_SK_XML_DOCUMENT	x									
4		CYBS_SK_ERROR_INFO			x	x	x	x	x	x	x	x
5		CYBS_SK_RAW_REPLY					x	x				x
6		CYBS_SK_FAULT_DOCUMENT							x	x	x	
7		CYBS_SK_FAULT_CODE							x	x	x	
8		CYBS_SK_FAULT_STRING							x	x	x	
9		CYBS_SK_FAULT_REQUEST_ID							x			

Using Name-Value Pairs

This section explains how to use the client to request CyberSource services by using name-value pairs.

Requesting CyberSource Services

To request CyberSource services, write code that:

- Collects information for the CyberSource services that you will use
- Assembles the order information into requests
- Sends the requests to the CyberSource server

Important The CyberSource servers do not support persistent HTTP connections.

- Processes the reply information

The instructions in this section explain how to use PHP to request CyberSource services. For a list of API fields to use in your requests, see "Related Documents," page 21.

Creating and Sending the Request

Note The code in this section's example is incomplete. For a complete sample program, see the `authCaptureSample.php` file in the `<installation directory>/samples/nvp` directory, or see the sample PHP pages.

To use any CyberSource service, you must create and send a request that includes the required information for that service.

The following example shows the basic PHP code for requesting CyberSource services. In this example, Jane Smith is buying an item for $29.95.

Loading the Configuration Settings

First load the configuration settings from a file:

```
$config = cybs_load_config( 'cybs.ini' );
```

You could instead create an array and add each configuration setting separately. You could also use a combination of the two methods: You could read the settings from a file and then add new settings dynamically with the array to override the settings read from the file.

Creating an Empty Request Array

You next create an array to hold the request fields:

```
$request = array();
```

Adding the Merchant ID

You next add the CyberSource merchant ID to the request. You can let the CyberSource PHP extension automatically retrieve the merchant ID from the `$config` array, or you can set it directly in the `$request` array (see below). The `$request` array value overrides the `$config` array value.

```
$request['merchantID'] = 'infodev';
```

Adding Services to the Request Array

You next indicate the service you want to use by adding the field to the request. For example, to request a credit card authorization:

```
$request['ccAuthService_run'] = 'true';
```

Requesting a Sale

You can request multiple services by adding additional fields to the request. For example, if you fulfill the order immediately, you can request credit card authorization and capture together (referred to as a “sale”):

```
$request['ccAuthService_run'] = 'true';
$request['ccCaptureService_run'] = 'true';
```

Adding Service-Specific Fields to the Request Array

You next add the fields that are used by the services that you are requesting. If you request multiple services and they share common fields, you must add the field once only.

```
$request['merchantReferenceCode'] = '3009AF229L7W';
$request['billTo_firstName'] = 'Jane';
$request['billTo_lastName'] = 'Smith';
$request['card_accountNumber'] = '4111111111111111';
$request['item_0_unitPrice'] = '29.95';
```

The example above shows only a partial list of the fields you must send. Refer to "Related Documents," page 21 for information about the guides that list all of the fields for the services that you are requesting.

Sending the Request

You next create the array that will hold the reply and send the request:

```
$reply = array();
$status = cybs_run_transaction( $config, $request, $reply );
```

Interpreting the Reply

Handling the Return Status

The `$status` value is the handle returned by the cybs_run_transaction() method. The `$status` indicates whether the CyberSource server received the request, the client received the reply, or there were any errors or faults during transmission. See "Possible Return Status Values," page 362 for descriptions of each status value. For an example in addition to the following one, see the `authCaptureSample.php` file in the *<installation directory>*`/samples/nvp` directory.

```
if ($status == 0)
  // Read the value of the "decision" in the $reply array.
  $decision = $reply['decision'];
  // If decision=ACCEPT, indicate to the customer that the request was successful.
  // If decision=REJECT, indicate to the customer that the order was not approved.
  // If decision=ERROR, indicate to the customer that an error occurred and to try
  // again later.
  // Now get reason code results:
  // $strContent = getReplyContent( $reply);
  // See "Processing the Reason Codes," page 371 for how to process the
  // reasonCode from the reply.
  // Note that getReplyContent() is included in this document to help you
  // understand how to process reason codes, but it is not included as part of the
  // sample scripts or sample PHP pages.

else
{
handleError( $status, $request, $reply );
}
//----------------------
function handleError( $status, $request, $reply )
//----------------------
  // handleError() shows how to handle the different errors that can occur.
{
  switch ($status)
  {
    // There was a problem with the parameters passed to cybs_run_transaction()
    case CYBS_S_PHP_PARAM_ERROR:
      // Non-critical error.
      // Tell customer the order could not be completed and to try again later.
      // Notify appropriate internal resources of the error.
      break;

    // An error occurred before the request could be sent.
    case CYBS_S_PRE_SEND_ERROR:
      // Non-critical error.
      // Tell customer the order could not be completed and to try again later.
      // Notify appropriate internal resources of the error.
      break;

    // An error occurred while sending the request.
    case CYBS_S_SEND_ERROR:
      // Non-critical error.
      // Tell customer the order could not be completed and to try again later.
      break;
    // An error occurred while waiting for or retrieving the reply.
    case CYBS_S_RECEIVE_ERROR:
      // Critial error.
      // Tell customer the order cannot be completed and to try again later.
      // Notify appropriate internal resources of the error.
      // See the sample code for more information about handling critical errors.
      break;

    // An error occurred after receiving and during processing of the reply.
    case CYBS_S_POST_RECEIVE_ERROR:
      // Critical error.
      // Tell customer the order could not be completed and to try again later.
      // Look at CYBS_SK_RAW_REPLY in $reply for the raw reply.
      // Notify appropriate internal resources of the error.
      // See the sample code for more information about handling critical errors.
      break;

    // CriticalServerError fault
    case CYBS_S_CRITICAL_SERVER_FAULT:
      // Critial error.
      // Tell customer the order could not be completed and to try again later.
      // Read the various fault details from the $reply.
      // Notify appropriate internal resources of the fault.
      // See the sample code for more information about reading fault details and
      // handling a critical error.
      break;
    // ServerError fault
    case CYBS_S_SERVER_FAULT:
      // Non-critical error.
      // Tell customer the order could not be completed and to try again later.
      // Read the various fault details from the $reply.
      // See the sample code for information about reading fault details.
      break;

    // Other fault
    case CYBS_S_OTHER_FAULT:
      // Non-critical error.
      // Tell customer the order could not be completed and to try again later.
      // Read the various fault details from the $reply.
      // Notify appropriate internal resources of the fault.
      // See the sample code for information about reading fault details.
      break;
    // HTTP error
    Case CYBS_S_HTTP_ERROR:
      // Non-critical error.
      // Tell customer the order cannot be completed and to try again later.
      // Look at CYBS_SK_RAW_REPLY in $reply for the raw reply.
      break;
  }
}
```

Processing the Reason Codes

After the CyberSource server processes your request, it sends a reply message that contains information about the services you requested. You receive different fields depending on the services you request and the outcome of each service.

To use the reply information, you must integrate it into your system and any other system that uses that data. For example, you can store the reply information in a database and send it to other back office applications.

You must write an error handler to process the reply information that you receive from CyberSource. Do not show the reply information directly to customers. Instead, present an appropriate response that tells customers the result.

Important

Because CyberSource may add reply fields and reason codes at any time, you should parse the reply data according to the names of the fields instead of their order in the reply.

The most important reply fields to evaluate are the following:

- **decision**: A one-word description of the results of your request. The decision is one of the following:
  - `ACCEPT` if the request succeeded
  - `REJECT` if one or more of the services in the request was declined
  - `REVIEW` if you are a CyberSource Advanced merchant using CyberSource Decision Manager and it flags the order for review. See "Handling Decision Manager Reviews," page 373 for more information.
  - `ERROR` if there was a system error. See "Retrying When System Errors Occur," page 375 for important information about handling system errors.
- **reasonCode**: A numeric code that provides more specific information about the results of your request.

You also receive a reason code for each service in your request. You can use these reason codes to determine whether a specific service succeeded or failed. If a service fails, other services in your request may not run. For example, if you request a credit card

authorization and capture, and the authorization fails, the capture does not run. The reason codes for each service are described in the *Credit Card Services User Guide* for CyberSource Essentials merchants or in the service developer guide for CyberSource Advanced merchants.

Important CyberSource reserves the right to add new reason codes at any time. If your error handler receives a reason code that it does not recognize, it should use the decision to interpret the reply.

The following is an example:

```
// Note that getReplyContent() is included in this document to help you understand
// how to process reason codes, but it is not included as part of the sample scripts
// or sample PHP pages.
//----------------
function getReplyContent( $reply )
//----------------
{
  $reasonCode = $reply['reasonCode']
  switch ($reasonCode)
  {
    // Success
    case '100':
      return( sprintf(
        "Request ID: %s\nAuthorizedAmount: %s\nAuthorization Code: %s,
        $reply['requestID'], $reply['ccAuthReply_amount'],
        $reply['ccAuthReply_authorizationCode'] ) );
        break;
    // Insufficient funds
    case '204':
      return( sprintf(
      "Insufficient funds in account. Please use a different card or select another
         form of payment." ) );
      break;

    // Add other reason codes here that you must handle specifically. For all
    // other reason codes, return an empty string, in which case, you should
    // display a generic message appropriate to the decision value you received.
    default:
      return ( '' );
  }
}
```

Handling Decision Manager Reviews

If you use CyberSource Decision Manager, you may also receive the `REVIEW` value in the **decision** field. `REVIEW` means that Decision Manager has marked the order for review based on how you configured the Decision Manager rules.

If you will be using Decision Manager, you have to determine how to handle the new `REVIEW` value. Ideally, you will update your order management system to recognize the `REVIEW` response and handle it according to your business rules. If you cannot update your system to handle the `REVIEW` response, CyberSource recommends that you choose one of these options:

- If you authorize and capture the credit card payment at the same time, treat the `REVIEW` response like a `REJECT` response. Rejecting any orders that are marked for review may be appropriate if your product is a software download or access to a Web site. If supported by your processor, you may also want to reverse the authorization.
- If you approve the order after reviewing it, convert the order status to `ACCEPT` in your order management system. You can request the credit card capture without requesting a new authorization.
- If you approve the order after reviewing it but cannot convert the order status to `ACCEPT` in your system, request a new authorization for the order. When processing this new authorization, you must disable Decision Manager. Otherwise the order will be marked for review again. For details about the API field that disables Decision Manager, see the *Decision Manager Developer Guide Using the Simple Order API* (PDF | HTML) or the *Decision Manager Developer Guide Using the SCMP Order API* (PDF | HTML).

Alternately, you can specify a custom business rule in Decision Manager so that authorizations originating from a particular internal IP address at your company are automatically accepted.

If supported by your processor, you may want to reverse the original authorization.

Requesting Multiple Services

When you request multiple services in one request, CyberSource processes the services in a specific order. If a service fails, CyberSource does not process the subsequent services in the request.

For example, in the case of a sale (a credit card authorization and a capture requested together), if the authorization service fails, CyberSource will not process the capture service. The reply you receive only includes reply fields for the authorization.

This following additional example applies **to CyberSource Advanced merchants only.**

Many CyberSource services include “ignore” fields that tell CyberSource to ignore the result from the first service when deciding whether to run the subsequent services. In the case of the sale, even though the issuing bank gives you an authorization code, CyberSource might decline the authorization based on the AVS or card verification results. Depending on your business needs, you might choose to capture these types of declined authorizations anyway. You can set the **businessRules_ignoreAVSResult** field to “true” in your combined authorization and capture request:

```
$request['businessRules_ignoreAVSResult'] = 'true';
```

This tells CyberSource to continue processing the capture even if the AVS result causes CyberSource to decline the authorization. In this case you would then get reply fields for both the authorization and the capture in your reply.

Note You are charged only for the services that CyberSource performs.

Retrying When System Errors Occur

You must design your transaction management system to include a way to correctly handle CyberSource system errors. Depending on which payment processor is handling the transaction, the error may indicate a valid CyberSource system error, or it may indicate a processor rejection because of some type of invalid data. In either case, CyberSource recommends that you do not design your system to retry sending a transaction many times in the case of a system error.

Instead, CyberSource recommends that you retry sending the request only two or three times with successively longer periods of time between each retry. For example, after the first system error response, wait 30 seconds and then retry sending the request. If you receive the same error a second time, wait one minute before you send the request again. Depending on the situation, you may decide you can retry sending the request after a longer time period. Determine what is most appropriate for your business situation.

If after several retry attempts you are still receiving a system error, it is possible that the error is actually being caused by a processor rejection and not a CyberSource system error. In that case, we suggest that you either:

- Search for the transaction in the Business Center, look at the description of the error on the Transaction Detail page, and call your processor to determine if and why they are rejecting the transaction.
- Contact CyberSource Customer Support to confirm whether your error is truly caused by a CyberSource system issue.

If TSYS Acquiring Solutions is your processor, you may want to follow the first suggestion as there are several common TSYS Acquiring Solutions processor responses that are returned to you as system errors and that only TSYS Acquiring Solutions can address.

Using XML

This section describes how to request CyberSource services using XML.

Requesting CyberSource Services

To request CyberSource services, write code that:

- Collects information for the CyberSource services that you will use
- Assembles the order information into requests
- Sends the requests to the CyberSource server

> **Important** The CyberSource servers do not support persistent HTTP connections.

- Processes the reply information

The instructions in this section explain how to write the code that requests these services. For a list of API fields to use in your requests, see "Related Documents," page 21.

Sample Code

We suggest that you examine the name-value pair sample code provided in `authCaptureSample.php` before implementing your code to process XML requests. The sample will give you a basic understanding of how to request CyberSource services. The sample code file is located in the `<installation directory>/samples/nvp` directory.

After examining that sample code, read this section to understand how to create code to process XML requests. Note that the code in this section's example is incomplete. For a complete sample program, see the `authSample.php` file in the `<installation directory>/samples/xml` directory.

Creating a Request Document

The client allows you to create an XML request document using any application, then send the request to CyberSource. For example, if you have a customer relationship management (CRM) system that uses XML to communicate with other systems, you can use the CRM system to generate request documents.

The request document must validate against the XML schema for CyberSource transactions. To view the schema, go to
https://ics2wsa.ic3.com/commerce/1.x/transactionProcessor
and look at the XSD file for the version of the Simple Order API you are using.

Important Make sure that the elements in your document appear in the correct order. If they do not, your document will not validate, and your request will fail.

The example developed in the following sections shows a basic XML document for requesting CyberSource services. In this example, Jane Smith is buying an item for $29.95.

The XML document in this example is incomplete. For a complete example, see the `auth.xml` document in the `samples/xml` directory.

Creating an Empty Request

Add the XML declaration and the document’s root element:

```
<?xml version="1.0" encoding="utf-8"?>
<requestMessage xmlns="urn:schemas-cybersource-com:transaction-data-
1.18">
</requestMessage>
```

When you construct a request, you must indicate the correct namespace for the elements, and the namespace must use the same API version that you specify in the configuration settings file. For example, if `targetAPIVersion=1.18` in the `cybs.ini` file, the namespace must be `urn:schemas-cybersource-com:transaction-data-1.18`.

Note The XML document that you receive in the reply always uses a prefix of `c:` (for example, `xmlns:c="urn:schemas-cybersource-com:transaction-data-1.18"`). Make sure you use an XML parser that supports namespaces.

Adding the Merchant ID

You next add the CyberSource merchant ID to the request.

Note If you specify a merchant ID in the XML document, it overrides the merchant ID you specify in the configuration settings file.

```
<?xml version="1.0" encoding="utf-8"?>
<requestMessage xmlns="urn:schemas-cybersource-com:transaction-data-
1.18">
   <merchantID>infodev</merchantID>
</requestMessage>
```

Adding Services to the Request

You next indicate the service that you want to use by creating an element for that service in the request, then setting the element's `run` attribute to `true`. For example, to request a credit card authorization:

```
<?xml version="1.0" encoding="utf-8"?>
<requestMessage xmlns="urn:schemas-cybersource-com:transaction-data-
1.18">
  <merchantID>infodev</merchantID>
  <ccAuthService run="true"/>
</requestMessage>
```

Requesting a Sale

You can request multiple services by adding additional elements. For example, if you fulfill the order immediately, you can request a credit card authorization and capture together (referred to as a "sale"):

```
<?xml version="1.0" encoding="utf-8"?>
<requestMessage xmlns="urn:schemas-cybersource-com:transaction-data-
1.18">
  <merchantID>infodev</merchantID>
  <ccAuthService run="true"/>
  <ccCaptureService run="true"/>
</requestMessage>
```

Adding Service-Specific Fields to the Request

You next add the fields that are used by the services you are requesting. Most fields are child elements of container elements; for example, a `<card>` element contains the customer's credit card information.

```
<?xml version="1.0" encoding="utf-8"?>
<requestMessage xmlns="urn:schemas-cybersource-com:transaction-data-
1.18">
  <merchantID>infodev</merchantID>
  <billTo>
    <firstName>Jane</firstName>
    <lastName>Smith</lastName>
  </billTo>
  <item id="0">
    <unitPrice>29.95</unitPrice>
  </item>
  <card>
    <accountNumber>4111111111111111</accountNumber>
  </card>
  <ccAuthService run="true"/>
</requestMessage>
```

The example above shows only a partial list of the fields you must send. Refer to "Related Documents," page 21 for information about the guides that list all of the fields for the services that you are requesting.

Sending the Request

Once you have created an XML document, you use PHP to send the request to CyberSource.

Loading the Configuration Settings

First load the configuration settings from a file:

```
$config = cybs_load_config( 'cybs.ini' );
```

Note

The namespace that you specify in the XML document must use the same API version that you specify in the configuration settings file. For example, if `targetAPIVersion=1.18` in the file, the namespace must be `urn:schemas-cybersource-com:transaction-data-1.18`. The example code below retrieves the API version from the configuration settings file and places it in the XML document.

Reading the XML Document

```
// Read the XML document.
// See the authSample.php script for
// the implementation of getFileContent().
$inputXML = getFileContent( "MyXMLDocument.xml" );

// Retrieve the target API version from the configuration settings
// and replace the value in the XML document.
$inputXML
  = str_replace(
  "_APIVERSION_", $config[CYBS_C_TARGET_API_VERSION], $inputXML );
```

Sending the Request

You next create the request array, add the XML document to the array, and send the request:

```
$request = array();
$request[CYBS_SK_XML_DOCUMENT] = $inputXML;

// send request
$reply = array();
$status = cybs_run_transaction( $config, $request, $reply );
```

Interpreting the Reply

Handling the Return Status

The `$status` value is the handle returned by the cybs_run_transaction() method. The `$status` indicates whether the CyberSource server received the request, the client received the reply, or there were any errors or faults during transmission. See "Possible Return Status Values," page 362 for descriptions of each status value. For an example in addition to the following one, see the `authSample.php` file in the client's `<installation directory>/samples/xml` directory.

```
if ($status == CYBS_S_OK)
  // Read the value of the "decision" in the oReplyMessage.
  // This code assumes you have a method called getField ()
  // that retrieves the specified field from the XML document
  // in $reply[CYBS_SK_XML_DOCUMENT].
  $decision = getField( $reply, "decision" );
  // If decision=ACCEPT, indicate to the customer that
  // the request was successful.
  // If decision=REJECT, indicate to the customer that the
  ' order was not approved.
  ' If decision=ERROR, indicate to the customer that there
  // was an error and to try again later.
  ' Now get reason code results:
  // $strContent = getReplyContent( $reply );
  ' See "Processing the Reason Codes," page 371 for how to process the reasonCode
  ' from the reply.
  ' Note that getReplyContent() is included in this document to help you understand
  ' how to process reason codes, but it is not included as part of the sample
  ' scripts or sample PHP pages.

else {
handleError( $status, $request, $reply );
}
//---------------------
function handleError( $status, $request, $reply )
//---------------------
{
  switch ($status)
  {
    // There was a problem with the parameters passed to
    // cybs_run_transaction()
    case CYBS_S_PHP_PARAM_ERROR:
      // Non-critical error.
      // Tell customer the order could not be completed and to try again later.
      // Notify appropriate internal resources of the error.
      break;

    // An error occurred before the request could be sent.
    case CYBS_S_PRE_SEND_ERROR:
      // Non-critical error.
      // Tell customer the order could not be completed and to try again later.
      // Notify appropriate internal resources of the error.
      break;
    // An error occurred while sending the request.
    case CYBS_S_SEND_ERROR:
      // Non-critical error.
      // Tell customer the order could not be completed and to try again later.
      break;

    // An error occurred while waiting for or retrieving
    // the reply.
    case CYBS_S_RECEIVE_ERROR:
      // Critial error.
      // Tell customer the order could not be completed and to try again later.
      // Notify appropriate internal resources of the error.
      // See the sample code for more information about handling critical errors.
      break;

    // An error occurred after receiving and during processing
    // of the reply.
    case CYBS_S_POST_RECEIVE_ERROR:
      // Critical error.
      // Tell customer the order could not be completed and to try again later.
      // Look at CYBS_SK_RAW_REPLY in $reply for the raw reply.
      // Notify appropriate internal resources of the error.
      // See the sample code for more information about handling critical errors.
      break;
    // CriticalServerError fault
    case CYBS_S_CRITICAL_SERVER_FAULT:
      // Critial error.
      // Tell customer the order could not be completed and to try again later.
      // Read the various fault details from the $reply.
      // Notify appropriate internal resources of the fault.
      // See the sample code for more information about reading fault details and
      // handling a critical error.
      break;

    // ServerError fault
    case CYBS_S_SERVER_FAULT:
      // Non-critical error.
      // Tell customer the order could not be completed and to try again later.
      // Read the various fault details from the $reply.
      // See the sample code for information about reading fault details.
      break;
    // Other fault
    case CYBS_S_OTHER_FAULT:
      // Non-critical error.
      // Tell customer the order could not be completed and to try again later.
      // Read the various fault details from the $reply.
      // Notify appropriate internal resources of the fault.
      // See the sample code for information about reading fault details.
      break;

    // HTTP error
    Case CYBS_S_HTTP_ERROR:
      // Non-critical error.
      // Tell customer the order could not be completed and to try again later.
      // Look at CYBS_SK_RAW_REPLY in $reply for the raw reply.
      break;
  }
}
```

Processing the Reason Codes

After the CyberSource server processes your request, it sends a reply message that contains information about the services you requested. You receive different fields depending on the services you request and the outcome of each service.

To use the reply information, you must integrate it into your system and any other system that uses that data. For example, you can store the reply information in a database and send it to other back office applications.

You must write an error handler to process the reply information that you receive from CyberSource. Do not show the reply information directly to customers. Instead, present an appropriate response that tells customers the result.

Important

Because CyberSource may add reply fields and reason codes at any time, you should parse the reply data according to the names of the fields instead of their order in the reply. If your error handler receives a reason code that it does not recognize, it should use the decision to interpret the reply.

The most important reply fields to evaluate are the following:

- **decision**: A one-word description of the results of your request. The decision is one of the following:
    - `ACCEPT` if the request succeeded
    - `REJECT` if one or more of the services in the request was declined
    - `REVIEW` if you use CyberSource Decision Manager and it flags the order for review. See "Handling Decision Manager Reviews," page 385 for more information.

  - ERROR if there was a system error. See "Retrying When System Errors Occur," page 387 for important information about handling system errors.

- **reasonCode**: A numeric code that provides more specific information about the results of your request.

You also receive a reason code for each service in your request. You can use these reason codes to determine whether a specific service succeeded or failed. If a service fails, other services in your request may not run. For example, if you request a credit card authorization and capture, and the authorization fails, the capture does not run. The reason codes for each service are described in the *Credit Card Services User Guide* for CyberSource Essentials merchants or in the service developer guide for CyberSource Advanced merchants.

The following is an example:

```
// Note that getReplyContent() is included in this document to help you understand
// how to process reason codes, but it is not included as part of the sample
// scripts or sample PHP pages.
// This code assumes you have a method called getField() that retrieves the
// specified field from the XML document in $reply[CYBS_SK_XML_DOCUMENT].

//----------------

function getReplyContent( $reply )

//----------------

{
  $reasonCode = $reply['reasonCode']
  switch ($reasonCode)
  {
    // Success
    case '100':
      return( sprintf(
        "Request ID: %s\nAuthorizedAmount:
          %s\nAuthorization Code: %s,
        getField( $reply, 'requestID' ), getField ($reply,
          'ccAuthReply/amount' ),
        getField( $reply, 'ccAuthReply/authorizationCode' ) ) );
        break;

    // Insufficient funds
    case '204':
      return( sprintf(
      "Insufficient funds in account. Please use a different
        card or select another form of payment." ) );
      break;
    // add other reason codes here that you must handle specifically. For all
    // other reason codes, return an empty string, in which case, you should
    // display a generic message appropriate to the decision value you received.
    default:
      return ( '' );
  }
}
```

Handling Decision Manager Reviews

If you use CyberSource Decision Manager, you may also receive the `REVIEW` value in the **decision** field. `REVIEW` means that Decision Manager has marked the order for review based on how you configured the Decision Manager rules.

If you will be using Decision Manager, you have to determine how to handle the new `REVIEW` value. Ideally, you will update your order management system to recognize the `REVIEW` response and handle it according to your business rules. If you cannot update your system to handle the `REVIEW` response, CyberSource recommends that you choose one of these options:

- If you authorize and capture the credit card payment at the same time, treat the `REVIEW` response like a `REJECT` response. Rejecting any orders that are marked for review may be appropriate if your product is a software download or access to a Web site. If supported by your processor, you may also want to reverse the authorization.

- If you approve the order after reviewing it, convert the order status to `ACCEPT` in your order management system. You can request the credit card capture without requesting a new authorization.

- If you approve the order after reviewing it but cannot convert the order status to `ACCEPT` in your system, request a new authorization for the order. When processing this new authorization, you must disable Decision Manager. Otherwise the order will be marked for review again. For details about the API field that disables Decision Manager, see the *Decision Manager Developer Guide Using the Simple Order API* (PDF | HTML) or the *Decision Manager Developer Guide Using the SCMP Order API* (PDF | HTML).

Alternately, you can specify a custom business rule in Decision Manager so that authorizations originating from a particular internal IP address at your company are automatically accepted.

If supported by your processor, you may want to reverse the original authorization.

Requesting Multiple Services

When you request multiple services in one request, CyberSource processes the services in a specific order. If a service fails, CyberSource does not process the subsequent services in the request.

For example, in the case of a sale (a credit card authorization and a capture requested together), if the authorization service fails, CyberSource will not process the capture service. The reply you receive only includes reply fields for the authorization.

This following additional example applies **to CyberSource Advanced merchants only.**

Many CyberSource services include “ignore” fields that tell CyberSource to ignore the result from the first service when deciding whether to run the subsequent services. In the case of the sale, even though the issuing bank gives you an authorization code, CyberSource might decline the authorization based on the AVS or card verification results. Depending on your business needs, you might choose to capture these types of declined authorizations anyway. You can set the **businessRules_ignoreAVSResult** field to “true” in your combined authorization and capture request:

```
<businessRules>

    <ignoreAVSResult>true</ignoreAVSResult>

</businessRules>
```

This tells CyberSource to continue processing the capture even if the AVS result causes CyberSource to decline the authorization. In this case you would then get reply fields for both the authorization and the capture in your reply.

Note

You are charged only for the services that CyberSource performs.

Retrying When System Errors Occur

You must design your transaction management system to include a way to correctly handle CyberSource system errors. Depending on which payment processor is handling the transaction, the error may indicate a valid CyberSource system error, or it may indicate a processor rejection because of some type of invalid data. In either case, CyberSource recommends that you do not design your system to retry sending a transaction many times in the case of a system error.

Instead, CyberSource recommends that you retry sending the request only two or three times with successively longer periods of time between each retry. For example, after the first system error response, wait 30 seconds and then retry sending the request. If you receive the same error a second time, wait one minute before you send the request again. Depending on the situation, you may decide you can retry sending the request after a longer time period. Determine what is most appropriate for your business situation.

If after several retry attempts you are still receiving a system error, it is possible that the error is actually being caused by a processor rejection and not a CyberSource system error. In that case, we suggest that you either:

- Search for the transaction in the Business Center, look at the description of the error on the Transaction Detail page, and call your processor to determine if and why they are rejecting the transaction.

- Contact CyberSource Customer Support to confirm whether your error is truly caused by a CyberSource system issue.

If TSYS Acquiring Solutions is your processor, you may want to follow the first suggestion as there are several common TSYS Acquiring Solutions processor responses that are returned to you as system errors and that only TSYS Acquiring Solutions can address.

Advanced Configuration Settings

Using Alternate Server Configuration Settings

You use the serverURL and namespaceURI configuration settings if CyberSource changes the convention we use to specify the server URL and namespace URI, but we have not updated the client yet.

For example, these are the server URLs and namespace URI for accessing the CyberSource services using the Simple Order API version 1.18:

- Test server URL:

  `https://ics2wstesta.ic3.com/commerce/1.x/transactionProcessor`

- Production server URL:

  `https://ics2wsa.ic3.com/commerce/1.x/transactionProcessor`

- Namespace URI:

  `urn:schemas-cybersource-com:transaction-data-1.18`.

Note If you view the above URLs in a web browser, a list of the supported API versions and the associated schema files are displayed.

If in the future CyberSource changes these conventions, but does not provide a new version of the client, you can configure your existing client to use the new server and namespace conventions required by the CyberSource server.

Configuring Your Settings for Multiple Merchant IDs

If you have multiple merchant IDs, or if you are a reseller handling multiple merchants, you can have different configuration settings for different merchant IDs. You set these in the configuration object that you pass to the cybs_run_transaction() function. When using the samples provided in the client package, you set the configuration parameters in `cybs.ini` file.

All of the properties except merchantID can be prefixed with `<merchantID>.` to specify the settings for a specific merchant.

Example Merchant-Specific Properties Settings

If you have a merchant with merchant ID of `merchant123`, and you want enable logging only for that merchant, you can set the enableLog parameter to `true` for all requests that have `merchant123` as the merchant ID:

```
merchant123.enableLog=true
enableLog=false
```

The client disables logging for all other merchants.

APPENDIX

A

Using the Client Application Fields

This appendix lists optional client application fields that you can include in your request to describe your client application. Use these fields only if you are building an application to sell to others. For example, a shopping cart application. Do not use the fields if you are only integrating the client with your own web store.

Table 49 Client Application Fields

Field Name	Description	Data Type and Length
clientApplication	Application or integration that uses the client: for example: `ShoppingCart Pro` or `Web Commerce Server`. Do not include a version number.	String (50)
clientApplicationVersion	Version of the application or integration, for example: `5.0` or `1.7.3`.	String (50)
clientApplicationUser	User of the application or integration, for example: `jdoe`.	String (30)

If you use these fields in your request, you can view their values in the Transaction Search Details window of the Business Center.

Index